A HIGH-ACCURACY OPTICAL LINEAR ALGEBRA PROCESSOR FOR FINITE ELEMENT APPLICATIONS

FINAL REPORT ON

NASA Grant #NAG 1-409

PREPARED FOR

NASA Langley Research Center
Hampton, VA 23665

Contract Monitor: Dr. Carl J. Magee, Mail Stop 499

PREPARED BY

Carnegie-Mellon University
Department of Electrical and Computer Engineering
Pittsburgh, Pennsylvania 15213

Authors:
Dr. David Casasent (Principal Investigator)
and
Bradley K. Taylor

16 November 1984

A HIGH-ACCURACY OPTICAL LINEAR ALGEBRA PROCESSOR FOR FINITE ELEMENT APPLICATIONS

FINAL REPORT ON
NASA Grant #NAG 1-409

PREPARED FOR
NASA Langley Research Center
Hampton, VA 23665

Contract Monitor: Dr. Carl J. Magee, Mail Stop 499

PREPARED BY
Carnegie-Mellon University
Department of Electrical and Computer Engineering

Authors:
Dr. David Casasent (Principal Investigator)
and
Bradley K. Taylor

14 November 1984

Table of Contents

List of Figures

List of Tables

Abstract

Optical linear algebra processors are computationally efficient computers for solving matrix-matrix and matrix-vector oriented problems. Applications include missle guidance, Kalman filtering, linear-quadratic-regulators, and the solution of partial differential equations. Presently, the most substantially documented optical processors are analog. Optical system errors limit their dynamic range to 30-40 dB, which limits their accuracy to 0-12 bits. Large problems, such as the finite element problem in structural mechanics (with tens or hundreds of thousands of variables) which can exploit the speed of optical processors, require the 32-bit accuracy obtainable from digital machines. To obtain this required 32-bit accuracy with an optical processor, the data can be digitally encoded, thereby reducing the dynamic range requirements of the optical system (i.e., decreasing the effect of optical errors on the data), while providing increased accuracy.

This report describes a new digitally encoded optical linear algebra processor architecture for solving finite element and banded matrix-vector problems. A linear static plate bending case study is described which quantifies the processor requirements. Multiplication by digital convolution is explained, and the digitally encoded optical processor architecture is advanced. A banded matrix-vector multiplication implementation is described for the architecture, and a direct solution technique for solving finite element problems is detailed. Fabrication of the processor with existing components is described. The results of optical error simulations for the processor implementation of selected multiplications are described. The dominant optical error sources are modelled in the simulation program, and the results demonstrate the effect of optical errors in a digitally encoded processor. Finally, future research topics are discussed, including optical error simulation of the case study solution, iterative algorithms, and data encoding in radices greater than two (binary).

1. Introduction

1.1 Introduction

Many scientific and engineering problems require various types of linear algebra calculations. The most basic and widely used linear algebra operations are vector inner products, vector outer products, matrix-vector products, and matrix-matrix products. These operations can be performed in many ways on standard digital computers. However, the sequential nature of digital processors does not exploit the inherent parallelism in linear algebra problems. For example, if the N multiplications required for an N-element vector inner product are performed in parallel rather than sequentially, the time needed is T_B vs. NT_B, or a savings of N-1 multiplication times, where T_B is the time required for one multiplication. There is a natural tendency and a practical need to make processing systems solve problems as fast as possible. This is driven by the large amount of computing time needed to solve many of today's significant engineering problems. Thus, the development of new parallel processing architectures for linear algebra problems has become an important task.

The most promising parallel processing architectures are optical processors. These systems represent numbers and values by light intensities and the modulation of light. A multiplication is performed when one beam of light at a certain intensity passes through a plane with a given transmittance. Optical systems can perform many multiplications in parallel and at high speed. With lenses, one can form sums of the products of many pairs of numbers. Many optical processors have been developed for the implementation of the linear algebra operations described above [31]. These processors are capable of data throughput rates much higher than those obtainable with digital processors.

Most of the optical processors developed so far represent each number with an analog signal in the processor. A major limitation of these processors is that they can only accurately represent numbers within a range of a few orders of

magnitude. In other words, their dynamic range is limited to 30-40 dB. A solution to this problem is to use encoded data in an optical processor. In this case, the numbers are encoded in binary or some other radix, and processed accordingly. This can be done at the expense of some speed and the size or complexity of the processor.

One application area that would benefit from such high-speed parallel linear algebra processors is finite element analysis. These problems require large dynamic range data and involve the solution of a large system of algebraic equations. In many cases, the matrices involved contain a few billion elements. Often, solutions to these problems require days of dedicated operation by a digital computer. Thus, the task is to solve a large set of linear algebraic equations quickly and accurately. For structural mechanics finite element problems, the matrix depends on the structure, and the right-hand side vector depends on the stucture loading. Generally, a solution with multiple right-hand sides is desired. Thus we initially consider direct algorithms for the solution of these problems.

Finite element problems have specific properties that can be exploited using a specialized processor. The goal of this research is to develop a digitally encoded optical linear algebra processor for solving finite element and other banded matrix-vector problems. Specifically, we will be concerned with finite element problems in structural mechanics. However, the processor will be developed to be general enough to perform many linear algebra operations for other matrix problems. Once developed, our optical linear algebra processor (OLAP) will be referred to simply as a high-accuracy OLAP.

1.2 Prior Work

A number of digitally encoded processing techniques have been developed. All of them involve using binary encoding only. Our work will address the issue of using other radices. In most proposed systems, the multiplication of two digitally encoded numbers is performed by a convolution of the bits of each

number. Some of the proposed systems achieve the multiplication with a vector outer product operation. Our OLAP uses the basic ideas from these architectures, but is unique in many ways, specifically in the data flow, hardware, convolution technique, encoding radix, and direct algorithm realization.

There has been some work by private groups in developing dedicated digital finite element machines. These processing systems are often composed of many processors working in parallel with the hardware designed for maximum data throughput, and usually oriented to implement a particular algorithm. There are some specialized algorithms that are very efficient when they are used with certain hardware configurations. Such a machine is being developed at NASA Langley Research Center in Hampton, Virginia [34].

Many algorithms have been developed for optical matrix-vector processors which are applicable for solving finite element problems [31]. These include both direct and iterative algorithms. One of these (LU decomposition) will be implemented on our OLAP for the research described in this report. It forms an upper triangular matrix from an N^{th} order matrix with N-1 matrix-matrix multiplications. Our OLAP will be suitable for other algorithms as well. Some iterative schemes possibly useful for solving finite element problems are also discussed.

1.3 Report Outline

This report begins in Chapter 2 with a description of the finite element method. The purpose of this chapter is to explain how finite element problems are developed, and their specific characteristics. A brief history of finite elements is included, and the fundamentals of the method are explained. An example of a finite element derivation is provided for a plane strain triangular finite element. The emphasis of this chapter, and this report, is linear static problems in structural mechanics. The problem formulation process, stiffness matrix assembly, and stiffness matrix properties are discussed in Chapter 2. Explanations of the various finite element equation solvers are given, and remarks on their advantages and

disadvantages are advanced. The last section of Chapter 2 discusses nonlinear and dynamic finite element problems and their solutions.

Our finite element case study problem to quantify our proposed OLAP's requirements is described in Chapter 3. It involves determining the plate bending behavior of an aluminum plate. The plate bending finite element is detailed and the structure discretization is described. An example of the assembly of one element in the model into the structure stiffness matrix is detailed. The characteristics of the matrix for this particular problem are then discussed.

Our proposed optical linear algebra processor is detailed in Chapter 4. The limitations of analog optical processors are discussed and then the multiplication of digitally encoded numbers by optical convolution is explained. The processor architecture and its operation are then detailed. Fabrication of the processing system using existing components is discussed. The data flow through the processor is detailed for solving banded matrix equations. A direct LU decomposition algorithm is described for solving finite element problems. Substructuring in finite element problem formulation and matrix partitioning are discussed. It is shown how only one channel of our processor is needed to implement the LU decomposition algorithm. Hence, it is quite new and most attractive.

The digital computer simulation of the optical processor is described in Chapter 5. The significant error sources in the processor are discussed, and their modelling is detailed. The simulation software components are then described, and some general remarks are advanced about the error mechanisms in the processor. The results of our error simulations on the multiplication of three sets of two numbers are given, and the performance of the OLAP with optical errors is evaluated.

Finally, the conclusion and a summary of this research are given in Chapter 6. The significant portions of the report are reviewed. Future work is discussed.

This includes optical error simulation of the case study solution to quantify the processor performance, iterative algorithms, encoding in other radixes, nonlinear and dynamic problems, and twos complement representation.

1.4 Contributions

This report contains some new, significant concepts in the area of optical linear algebra processing. Perhaps the most important is the digitally encoded processor architecture introduced in Chapter 4. It is unique in its data flow for banded matrix problems, and the method used to perform the convolution of the digital bits. The shift-and-add method of multiplying binary numbers is used, with the shift-and-add's being performed in the detector plane. This architecture easily partitions to allow processing of larger bandwidth matrices.

The concept of encoding in radices other than two (binary) is also discussed. Although most of the report is concerned with binary encoding, encoding in other radices is discussed in significant places. This has the potential of increasing processor speed and decreasing processor size, while still yielding accurate results.

Another new concept introduced is that of a 1-channel processor architecture capable of implementing the LU decompositon algorithm with our banded matrix-vector product realization. The implementation guarantees that only two inputs are used at any time, and one of them is known. The contribution of the known input (which is always a 1) can be hardwired in the architecture, and thus only one processor channel is needed for the variable input.

An optical error simulation program was developed, and error simulations were performed for three sets of 32-bit multiplications on our optical linear algebra processor. This is the first optical system error simulation performed for a multi-purpose digitally encoded optical processor, and it provided some very useful results. The simulations showed how optical errors affect the performance of a digitally encoded processor.

1.5 Notation

This section is intended to clarify the notation used in this report. The first issue to be considered is the definition of dB that is used throughout the report. Optical processors compute with light, and the intensity, rather than the amplitude, of the light is detected by all detector systems. The operations performed in our optical processor will be entirely proportional to intensity. The conventional dB definition for comparison of amplitudes A_1 and A_2 is

$$20LOG_{10}(A_1/A_2) = dB \tag{1.1}$$

However, since intensity is amplitude squared, the dB definition we will use is

$$10LOG_{10}(I_1/I_2) = dB \tag{1.2}$$

where I_1 and I_2 are the two intensities being compared. The definition in (1.2) is used for all dB values given in this report.

Clarification needs to be made concerning the matrix and vector notation used in this report. Chapters 2 and 3 deal with structural mechanics, and Chapters 4 and 5 deal with linear algebra. The standard matrix-vector notation used in these two disciplines is very different. We feel that for continuity reasons, it is important to use the conventional notation for each topic area in the appropriate parts of the text. Thus, two types of matrix-vector notations are used, one type in Chapters 2 and 3 (for the exclusive structural mechanics information), and standard linear algebra notation in the remaining chapters.

Both notations are extremely simple, and the presence of both should not confuse the reader. In the structural mechanics literature, letters representing a matrix or vector are explicitly shown within some type of bracket, the type of bracket indicating a matrix or a vector. In standard linear algebra literature (including optical processing), a matrix is denoted by a boldface or underlined upper-case letter, and a vector is denoted by a boldface or underlined lower-case letter. In this report, boldface will be used rather than underlining.

Thus, in Chapters 2 and 3, a matrix indexed by the letter Z is denoted by

[Z]

and a vector indexed by the letter z is denoted by

{z}

In this notation the matrix or vector is defined by the type of bracket only, and not the type of letter. To make the notation conversion clearer, all matrices are also upper-case letters, and all vectors are also lower-case letters in Chapters 2 and 3.

In Chapters 4 and 5, a matrix indexed by the letter Z is denoted by

Z

and a vector indexed by the letter z is denoted by

z

By convention, square brackets enclosing a number denote a reference paper. These references can be found in the reference list in Chapter 7.

2. The Finite Element Method

2.1 Introduction

This chapter contains a concise review of the finite element method, and it emphasizes aspects of the method that are of particular interest. A brief history (section 2.2), a discussion of fundamentals (section 2.3), and the derivation of the finite element equations (section 2.4) are given. The basic algorithm for solving finite element problems and its details are then presented (sections 2.5 - 2.7). Finally, nonlinear and dynamic problems are discussed (section 2.8). A specific finite element problem example is introduced in Chapter 3. Other problem examples may be found in [1], [2], and [3].

2.2 Finite Element Method Background

Many problems that engineers must solve concern the state or states of a continuum, whose behavior is governed by one or more partial differential equations. Examples of such problems are the electric field between two conductors, the stresses within a building during an earthquake, and the modes of vibration of an aircraft during flight. Exact solutions to the equations governing such problems are rare, requiring an approximate solution, if any, to the problem. Most approximate solution techniques, such as finite differences, series representation, and finite elements, require many algebraic operations, which poses a large computational problem. Before the advent of accessible digital computing in the 1960's, obtaining accurate approximate solutions was not feasible.

The finite element method is an approximation technique which developed rapidly once digital computing became available. It is an analytical procedure whose basic concept is that a continuum can be modelled analytically by dividing it into subdivisions. Each subdivision is modelled by a finite element, and the finite elements are connected together to model the entire continuum. The behavior of each finite element is described by a set of prescribed functions, which will be

called basis (or shape) functions. The basis functions will only guarantee a certain level of continuity across the continuum, however they will provide solutions that are satisfactory approximations to the actual behavior of the continuum.

The finite element method results in a large system of algebraic equations. An advantage of choosing finite elements over other approximate solution techniques is that the equation formulation is extremely appropriate for implementation on a computer. Another advantage is that the finite element method can be used to analyze complicated and irregular continuums subject to difficult loading conditions. The most important advantage is that the finite element method performs very well when properly used, yielding accurate results.

There is a wide variety of applications for the finite element method. Most of the finite element work originated and is still applied in structural mechanics. Application of finite elements for determining electric and magnetic fields in semiconductors and power distribution systems has become very popular. The technique is also used in fluid flow and heat flow analysis.

The specific behavior of a continuum can be linear or nonlinear. The choice of using linear or nonlinear finite element analysis depends on which approximation best matches the actual behavior, and what type of anaylsis is computationally feasible. The loading on the continuum (its environment) can be static, fixed loads, or dynamic, time-varying loads. The simplest finite element analysis case is a linear approximation with static loading, or a linear static problem. These problems require solution of only one set of linear algebraic equations. Both nonlinear and dynamic problems greatly complicate the analysis and solution. These problems will be discussed in section 2.8. All other sections of this chapter and the majority of this report will consider only linear static finite element problems in structural mechanics.

2.3 Finite Element Fundamentals

The subdivision of a continuous structure for finite element analysis is called discretization. Each discrete part of the structure is modelled by a finite element. Each finite element is permitted a prescribed behavior, which depends on the type of structural unit being modelled by that element. The shape and interconnections of elements are defined by the nodes of each element. There are many types of finite elements. The simple bar element shown in Figure 2-1 consists of two nodes, one at each end of the bar.

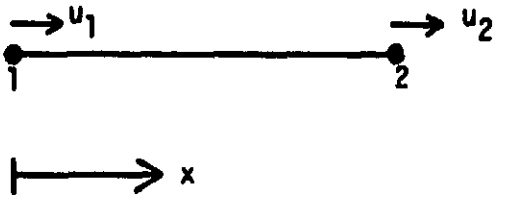


Figure 2-1: Simple Bar Finite Element

A quadrilateral element, shown in Figure 2-2, consists of a minimum of four nodes, one at each corner.

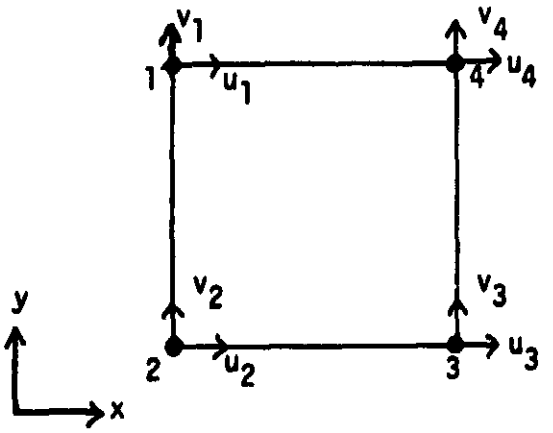


Figure 2-2: Quadrilateral Finite Element

All connections between elements in a model are made at the element's boundary nodes.

An element is allowed to deform according to the degrees of freedom (DOFs) defined at its nodes. One or more DOFs may be defined at every node of a finite element. If a finite element model consists of n nodes and d DOFs per node, then there are n times d DOFs in that finite element problem. The DOFs are usually displacements, but may also be derivatives of displacements (rotations), stresses, forces, or some other defined quantity. For the bar of Figure 2-1, a single DOF u is defined at both nodes; it is the displacement in the x direction. The quadrilateral element of Figure 2-2 has displacements u and v in the x and y directions as its DOFs.

Some terminology clarification should be made here. In strict terms, a variable is actually defined for each DOF allowed at each node. For example, if the DOF is movement along a rectangular coordinate axis, the variable is a displacement in that direction. The variable takes on the value of how much a node displaced; it is not proper to say the DOF takes on that displacement value, the DOF is always the same defined movement. However, we will not adhere to the proper terminology (nobody ever does), and will usually refer to the variables and their numerical values as the DOFs defined for them. Thus, "solving for the DOF" will mean the same as solving for the numerical value of the variable defined for that DOF. This point will become clear and insignificant as the reader continues.

Another subtle point should be made here. In most of the structural mechanics finite element literature, the term displacements is used to refer to all the variable types, even if some are rotations, stresses, etc. This is done for convenience, but it sometimes hides the fact that more than one type of DOF can be defined in a problem. When it is appropriate and less confusing, the term displacements will be used in this report in such a manner. Mostly, however, we will refer to the DOFs, which is considerably more general.

A set of prescribed functions determine the behavior, or deformation, of an

element. These functions are called shape, or basis functions. There is one basis function, defined over the element, for each DOF defined at the nodes of an element. The finite element method is unique with respect to other matrix structure analysis procedures in that the basis functions are interpolatory. That is, the basis function for a particular DOF takes on a value of one at the node at which that DOF is defined, and a value of zero at all other nodes on the element. The basis functions are usually polynomials, which will only be considered here, but other functions could be used. Thus, the basis functions between nodes are defined by polynomials whose order is dictated by the number of DOFs on the element. The DOF values define the coefficients of the polynomials. The simple bar element has two DOFs, thus its basis functions are linear in x, since two constants define a line.

Each DOF value, for a given dimension on the element (x, y, etc.), is multiplied by its corresponding basis function. The resulting functions are added to determine an equation governing the deformation of the element in that dimension. If the element is multi-dimensional, the governing equations for each dimension are multiplied together to yield a general deformation equation for any point on the element.

For a one-dimensional example, consider the bar of Figure 2-3 [1]. It is fixed at one end and loaded by force P and distributed load q. The bar is discretized into three simple bar elements as shown in Figure 2-4. There is a single displacement u in the x direction defined for each node. The bar element is a linear element, thus the basis functions are linear functions of x. The basis function for each DOF is one at its defined node, and zero at the other node on the element. The three elements are connected, or assembled, together as in Figure 2-4. Nodes 2 and 3 are common to two elements, and thus the DOFs at those nodes are defined for both elements. Likewise, the basis functions for the DOFs at nodes 2 and 3 will be defined over both elements, and they will be piecewise linear. The basis function for node i is defined as $N_i(x)$, and $N_1(x)$ through $N_4(x)$ are shown, superimposed across all three elements, in Figure 2-5.

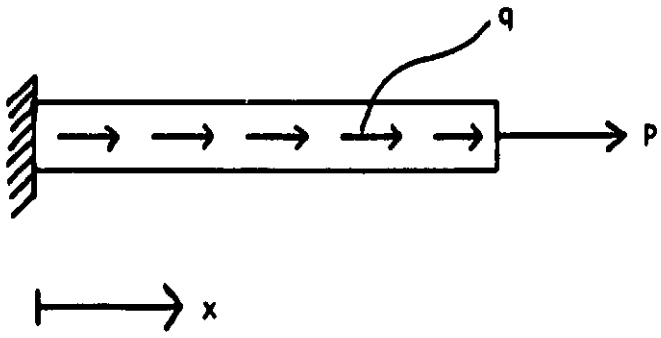


Figure 2-3: Loaded Bar Fixed at One End

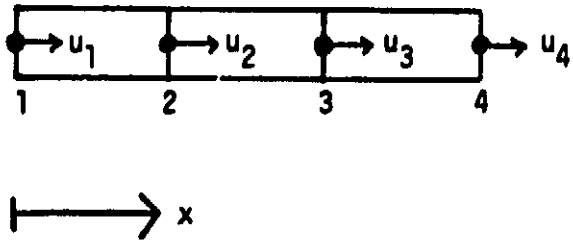


Figure 2-4: Discretized Bar - Three Elements

Once the four DOF values are solved for, they are multiplied by their corresponding basis functions; i.e., each basis function is weighted by its DOF value. The addition of the resulting functions across all three elements represents adding the contribution of each element to obtain the total displacement of the structure model. In more general terms, this is called element assembly. The result of adding the four weighted basis functions is an equation for the displacement of the bar across all three elements. Graphically, it results in a

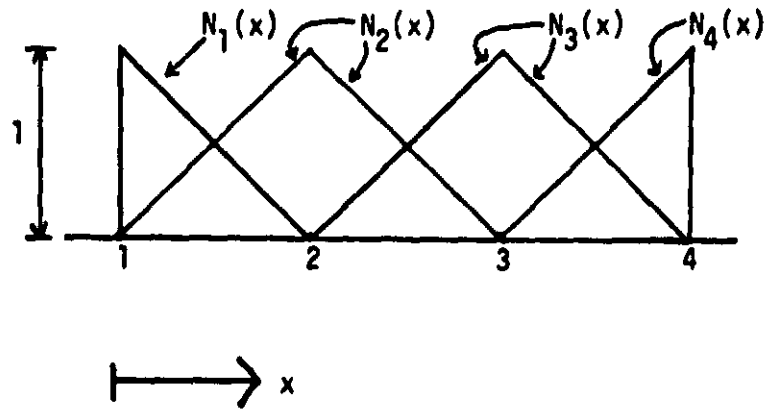


Figure 2-5: Basis Functions for Discretized Bar

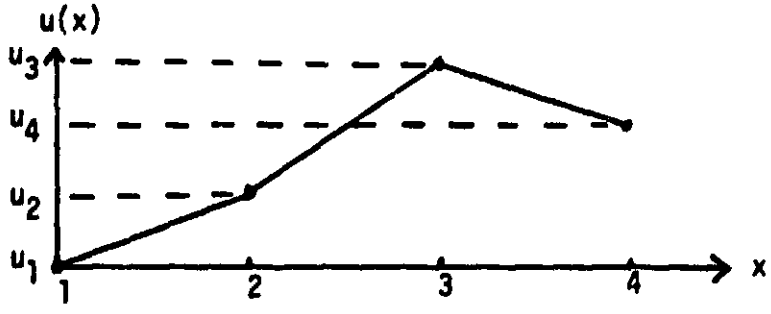


Figure 2-6: Finite Element Displacement Field

piecewise linear displacement field, which is shown in Figure 2-6. Note that u_1 at x=0 is zero. This is because that end of the bar is fixed, and that boundary condition was imposed on the problem. Boundary conditions are easily handled with the finite element method, and will be discussed in section 2.5.

The behavior obtainable by a finite element depends on its shape, the DOFs

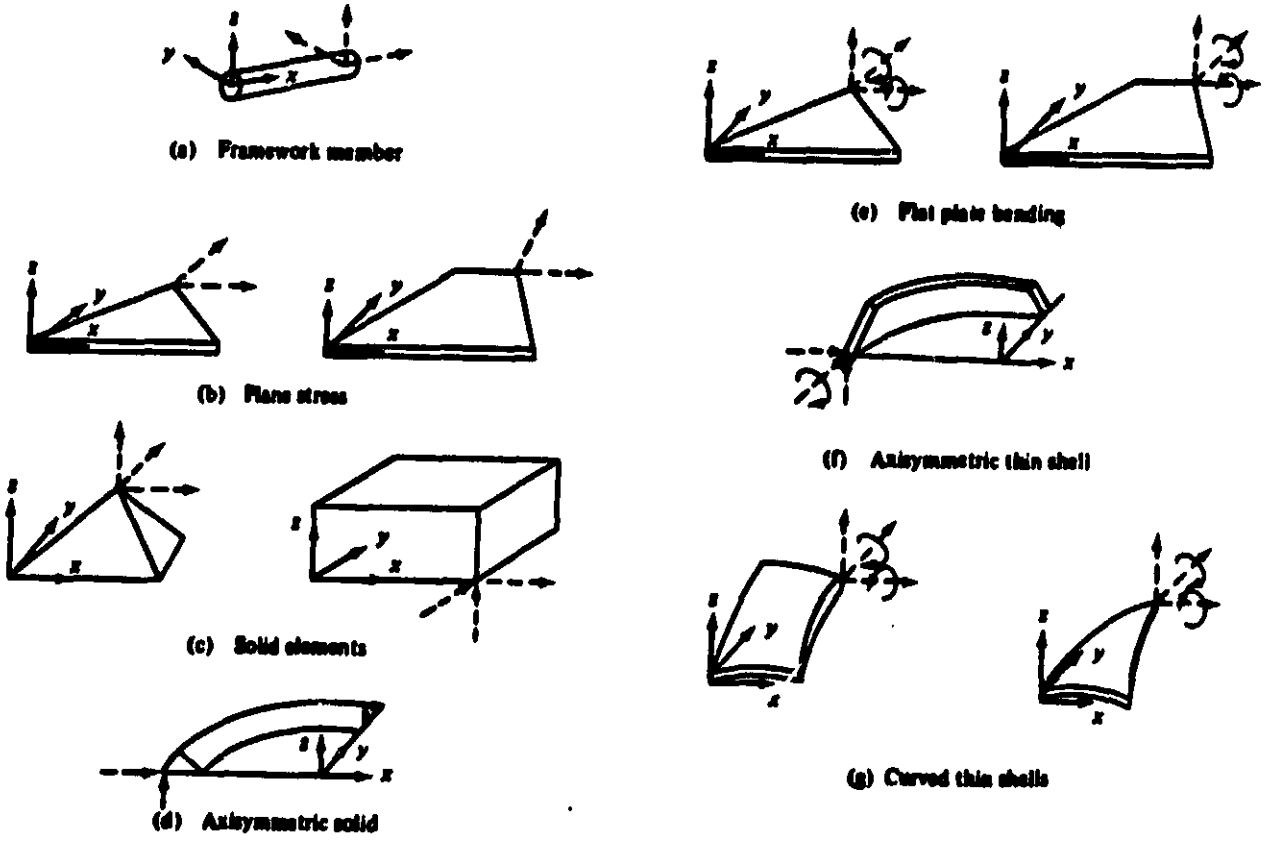
defined at its nodes, and the order of the basis functions, i.e., the order of the element. In general, higher order elements will provide a better approximation to the actual behavior of the structure being modelled. A variety of elements may be used to model structures. The choice of elements depends on the structure, loading, type of results being investigated, assumptions on the expected behavior of the structure, and other factors. These become complex decisions that are not appropriate for discussion here; such information is available in [1], [2], and [3].

Some basic types of finite elements are shown in Figure 2-7 [2]. The coordinate axes and DOFs are shown for each element. Element a is a simple framework or beam element, and the elements in b are plane stress (or strain) triangles and quadrilaterals. They are the most common elements for two-dimensional analysis. The elements in c and d are three-dimensional solid elements. The elements in c are three-dimensional generalizations of the elements in b. The element in d is an axisymmetric element, which is very useful for modelling structures with symmetry about a central axis, such as pressure vessels, metal tanks, rotors, and shafts. The elements in e are more sophisticated plate bending elements. They are used to model the flat plate bending behavior of many structures, and will be used in the case study in Chapter 3. The elements in f and g are thin shell elements. They are used to describe the behavior of shell-like surfaces (airframes) by incorporating stretching and bending within the elements.

2.4 Derivation of Finite Element Equations

Every finite element problem can be described by a set of algebraic equations, expressed in matrix form. This is the most common description used, however other notations are possible, such as tensor notation. These equations may be derived through a variety of methods. The formulation of the finite element method spans such a wide range of theoretical topics that only a basic introduction will be given here, with an example of the derivation of a plane strain triangular finite element.

Figure 2-7: Common Finite Elements [2]



Finite element equation formulation techniques can be divided into two types [2], direct methods and variational methods. Direct methods are straightforward and allow insight into any limitations of the element formulation. Two popular direct methods are the direct method, which combines equations of equilibrium, strain-displacement equations, and constitutive relationships, and the method of weighted residuals, or Galerkin's method.

The variational methods are often called energy methods. These methods use calculus of variations and form element relationships by using equations related to the total work, or energy, in an element. Some variational methods are the principle of minimum potential energy, the principle of virtual work, and the principle of minimum complementary energy. Variational methods are usually preferred over direct methods because they can provide information on the convergence of an element, and can be used to formulate bounds on the numerical solution. An element is convergent if, as the finite element mesh is infinitely refined into more and smaller elements, the finite element solution approaches the actual solution to the problem. Regardless of which formulation method is used, they all, when properly applied, will yield the same finite element equations.

A popular formulation method is the principle of minimum potential energy. An example of a finite element formulation using this method will now be given for a triangular element undergoing plane strain deformation [4]. The formulation is given for an isotropic, linearly elastic material, as the minimum potential energy principle is only valid for elastic materials. The conditions of a body undergoing plane strain deformation are illustrated in Figure 2-8.

The quantities x, y, z are the coordinate axes, and u, v, w are the displacements in the x, y, and z directions, respectively. The plane strain deformation assumptions indicated in Figure 2-8 are: 1) f=f(x,y), the body force, f, and any boundary forces have a zero component in the z direction; 2) w=0, there is no displacement in the z direction; 3) u=u(x,y), v=v(x,y), the lateral

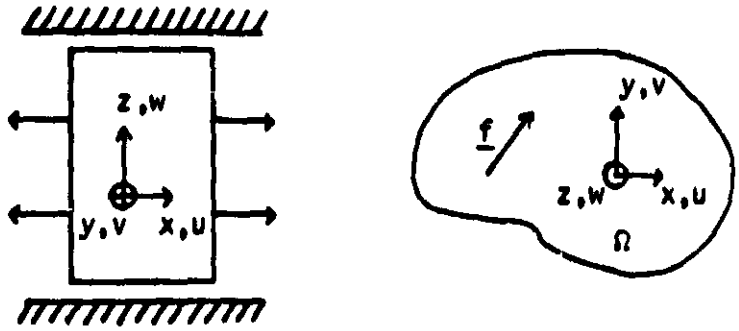


Figure 2-8: Plane Strain Deformation Conditions - Two Views

displacement fields are functions of x and y only. An example of a structure that can be modelled by plane strain deformation is a long dam, with the z-axis running along the length of the dam.

With the conditions given above, the nonvanishing strains are

$$\begin{aligned} \epsilon_x &= \partial u/\partial x \\ \epsilon_y &= \partial v/\partial y \\ \gamma_{xy} &= \partial u/\partial y + \partial v/\partial x = \gamma_{yx} \end{aligned} \tag{2.1}$$

which can be expressed in matrix form as the strain-displacement relationship

$$\begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ \partial/\partial y & \partial/\partial x \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.2}$$

or

$$\{\epsilon\} = [L]\{u'\} \qquad \text{where} \qquad \{u'\} = [u \;\; v]^T$$

For plane strain, where the σ's are stresses,

$$\epsilon_z = -(\nu/E)\sigma_x - (\nu/E)\sigma_y + (1/E)\sigma_z = 0 \tag{2.3}$$

From (2.3) and the generalized Hooke's Law, the stress-strain relationships can be written as

$$\begin{aligned} \epsilon_x &= [(1-\nu^2)/E]\sigma_x - [\nu(1+\nu)/E]\sigma_y \\ \epsilon_y &= -[\nu(1+\nu)/E]\sigma_x + [(1-\nu^2)/E]\sigma_y \\ \gamma_{xy} &= 2[(1+\nu)/E]\tau_{xy} \end{aligned} \qquad (2.4)$$

where ν is Poisson's ratio and E is Young's modulus. Writing (2.4) in matrix form and inverting yields

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = E/[(1+2\nu)(1+\nu)] \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & (1-2\nu)/2 \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix}$$

or

$$\{\sigma\} = [E]\{\epsilon\} \qquad (2.5)$$

where [E] is called the elasticity matrix

The potential energy Π (per unit length in the z direction, omitting initial stresses, strains, and surface tractions - Figure 2-8), where Ω is the cross-sectional body in x and y, is given by

$$\begin{aligned} \Pi &= 1/2 \int_\Omega (\sigma_x\epsilon_x+\sigma_y\epsilon_y+\tau_{xy}\gamma_{xy})dxdy - \int_\Omega (f_x u+f_y v)dxdy \\ &= 1/2 \int_\Omega \{\sigma\}^T\{\epsilon\}dxdy - \int_\Omega \{f\}^T\{u'\}dxdy \qquad (2.6) \end{aligned}$$

$$\text{where} \qquad \{f\} = [f_x \ f_y]^T$$

Substituting (2.5) into (2.6) and reversing the vector inner product in the second term, we obtain

$$\Pi = 1/2 \int_\Omega \{\epsilon\}^T[E]\{\epsilon\}dxdy - \int_\Omega \{u'\}^T\{f\}dxdy \qquad (2.7)$$

and substituting (2.2) into (2.7) yields

$$\Pi = 1/2 \int_{\Omega} \{u'\}^T[L]^T[E][L]\{u'\}dxdy - \int_{\Omega} \{u'\}^T\{f\}dxdy \qquad (2.8)$$

Now that an equation has been developed for the potential energy, the variational method can be used. The principle of minimum potential energy states: among all admissable displacements of a body, those which satisfy the essential (geometric) boundary conditions, the displacement that minimizes the potential energy Π is the stable solution of the governing equations of equilibrium and associated natural boundary conditions, and it is a global minimum for linearly elastic cases. Thus, if solved analytically, the displacement field obtained from the principle of minimum potential energy is guaranteed to be a solution to the partial differential equation governing the behavior of the body.

Ideally, one would like to analytically solve for an admissable function $\{u'(x,y)\}$ such that (2.8) is a minimum. Since it is usually impossible or extremely difficult to solve for the analytical solution, we discretize the variational principle with finite elements to obtain an approximate solution. Note that this is different than the approximation method of finite differences, where the differential equation is discretized. At this point the finite element discretization is introduced for $\{u'(x,y)\}$. Linear triangular elements will be used to discretize the body as shown in Figure 2-9. The body is modelled by many triangular elements, Ω_e, with one node at each vertex. The DOFs at each node are displacements u and v in the x and y directions, as indicated in Figure 2-9.

The basis function for each DOF at each node is a triangle that has a value of one at that node, and a value of zero at the other two nodes and along the entire opposite edge. The basis function $N_1(x,y)$ for node 1 is shown in Figure 2-10. The unshaded triangle is the finite element in the x-y plane, and the shaded triangle is the basis function for a DOF at node 1, and it has a value of

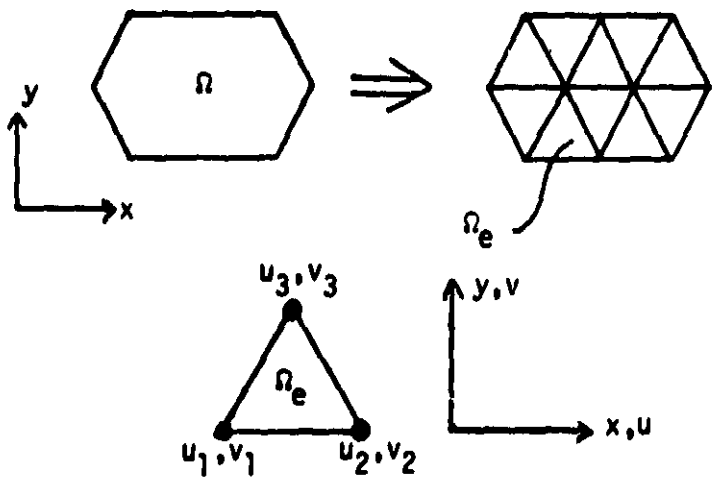


Figure 2-9: Finite Element Discretization

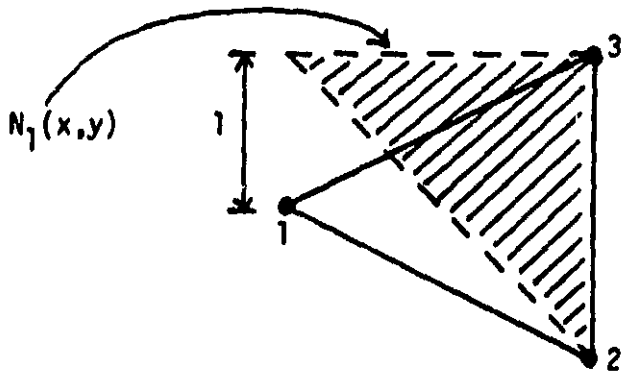


Figure 2-10: Basis Function for One Node of a Triangular Element

one there. The displacement function $\{u'(x,y)\}$ for one finite element is discretized in terms of the basis functions and nodal DOFs as follows

$$u'(x,y) = \begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix}$$

$$\text{or} \qquad \{u'(x,y)\} = [N(x,y)]\{d_e\} \tag{2.9}$$

where $\{d_e\}$ is the vector of nodal DOFs (see Figure 2-9). Substituting (2.2) into (2.9), the strains can be described as

$$\{\epsilon\} = [L][N(x,y)]\{d_e\}$$

or

$$\{\epsilon\} = [B(x,y)]\{d_e\} \tag{2.10}$$

where $[B(x,y)]=[L][N(x,y)]$.

The total potential energy of the discretized body is the sum of the potential energy of each finite element modelling the body. Substituting (2.10) into (2.7), the potential energy of a single element is given by

$$\begin{aligned} \Pi_e &= 1/2\{d_e\}^T \int_{\Omega_e} [B]^T[E][B]\{d_e\}dxdy \\ &\quad - \{d_e\}^T \int_{\Omega_e} [N]^T\{f\}dxdy \\ &= 1/2\{d_e\}^T[K_e]\{d_e\} - \{d_e\}^T\{r_e\} \end{aligned} \tag{2.11}$$

where

$$[K_e] = \int_{\Omega_e} [B]^T[E][B]dxdy$$

and

$$\{r_e\} = \int_{\Omega_e} [N]^T\{f\}dxdy \tag{2.12}$$

Minimizing Π_e with respect to the displacements $\{d_e\}$, i.e. setting $\delta\Pi_e=0$, results in the fundamental equation

$$[K_e]\{d_e\} - \{r_e\} = 0$$

or

$$[K_e]\{d_e\} = \{r_e\} \tag{2.13}$$

Equation 2.13 is the basic finite element equation governing the behavior of a single element. The matrix $[K_e]$ is called the elemental stiffness matrix, and it entirely defines the properties of a given element. The vector $\{d_e\}$ is the vector of nodal DOFs, and $\{r_e\}$ is the vector of equivalent elemental loads applied at the DOFs. The type of load depends on the DOF. For example, forces are the loads for displacements, moments are the loads for rotations, etc. In general, $[K_e]$ is a full matrix.

For the entire structure, or body, one wants to minimize the potential energy $\Pi=\sum_e \Pi_e$, the sum, over all elements, of the elemental potential energies. This process proceeds similarly to the derivation of (2.13), and results in the structural equation

$$[K]\{d\} = \{r\} \tag{2.14}$$

where $[K]=\sum_e [K_e]$, $\{r\}=\sum_e \{r_e\}$, and $\{d\}$ is a vector of all the DOFs in the structure. The matrix $[K]$ is called the structure stiffness (or just stiffness) matrix. It is formed by the proper summation, or assembly, of all the elemental stiffness matrices, a process which will be discussed in section 2.5. The vector $\{r\}$ is a vector of equivalent nodal loads on the structure at each DOF, corresponding exactly to the DOFs in the $\{d\}$ vector. Equivalent loads are defined as those loads needed to balance any distributed loads, initial stresses, or initial strains. Equivalent nodal loads are determined by (2.12), as are the stiffness matrix elements.

In a finite element problem, $[K]$ and $\{r\}$ are known, being assembled from (2.13). The unknowns are the elements of the $\{d\}$ vector, and we must solve a matrix equation to obtain those values. Assuming all the nodes in a problem have the same number of DOFs, n_1, and there are n_2 nodes, then there are $n=n_1 \times n_2$ DOFs in the problem. Equation 2.14 represents an n^{th} order system of linear

algebraic equations. The matrix [K] is n by n, and the vectors {d} and {r} are n by 1.

2.5 The Basic Finite Element Method Algorithm

Implementation of the finite element method for solving structural mechanics problems can be outlined in six general steps [5]:

1. Discretization of the structure; i.e., selection of elements interconnected at certain nodal points.
2. Evaluation of the element stiffness and equivalent elemental load matrices.
3. Assembly of the stiffness and equivalent nodal load matrices for the system of elements.
4. Introduction of the boundary conditions and external loads.
5. Solution of the resulting finite element system equations.
6. Calculations of strains and stresses based on the nodal DOF values.

Step 1 involves deciding what elements to use to best model the structure being investigated. Some knowledge or assumptions of the behavior of the structure and the desired results is needed here. The proper choice of elements to best model the geometry of the structure is often the most important decision made when analyzing a structure. The physical layout of the elements as they model the structure is just as important for obtaining good results. Typically, the bulk part of a structure is modelled with a uniform mesh of standard elements. Edges, corners, holes, and discontinuities in a structure may need to be modelled with many small elements. This is because displacement fields in a structure usually have a large variation in those areas, and a finer mesh of elements will obviously approximate such behavior better than one or two elements. An example is shown in Figure 2-11. The ability to make good decisions at this step of the process usually requires experience with finite element analysis.

It is appropriate to point out that different types and shapes of elements may be used together in a structure model, as long as they are compatible. Obviously, the DOFs at connecting nodes must be the same for any elements that

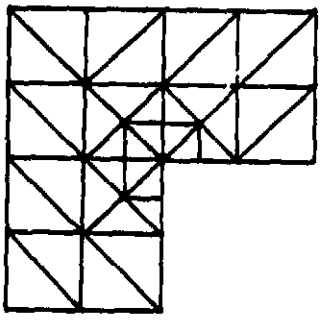


Figure 2-11: Triangular Finite Element Mesh Around a Corner

are connected. The conditions and assumptions on the behavior of elements must be compatible if different type elements are to be used in the same mesh. Otherwise, results will be quite unacceptable, if obtainable at all. Often, if one part of a structure exhibits a certain behavior, say plane strain, and an adjacent part undergoes some other behavior, a transition element is used to connect the elements modelling the different behaviors.

Step 2 is the formulation of the elemental stiffness matrix $[K_e]$ and the equivalent elemental load vector $\{r_e\}$, for each finite element in the model. The matrix and vector are formulated by (2.12) or other direct or variational methods. External loads are not applied to any element until the entire structure has been assembled. Thus, the non-zero values in the vector $\{r_e\}$ represent those loads needed to balance any distributed loads, initial stresses, or initial strains acting on the element.

For a given finite element type and shape, only a single elemental stiffness matrix needs to be derived. If needed, it can contain variables for the actual element size and material properties. Thus, (2.12) must only be evaluated once for each different finite element type in the model. Most models contain only a few different types of elements, if more than one.

Step 3 is the assembly of all the elemental equations, (2.13), into the

structure equation, (2.14). If each element has m DOFs, each system of elemental equations will be of m^{th} order. If there are n DOFs in the structure model, the system of structure equations will be of order n, and n is always greater than m for models with more than one element. Thus, the values in the elemental stiffness matrix and elemental equivalent load vector must be properly assembled into the larger structure stiffness matrix and equivalent nodal load vector. This operation is explained below. The procedure is simple, but difficult to explain without an example.

Consider a structure model with a total of n DOFs, with n nodes and one DOF defined at each node. The model is made up of some number of elements, with m nodes each and of course, one DOF per node. Equation 2.13 may be written as

$$\begin{aligned} k_{e11}d_{e1} + k_{e12}d_{e2} + \dots + k_{e1m}d_{em} &= r_{e1} \\ k_{e21}d_{e1} + k_{e22}d_{e2} + \dots + k_{e2m}d_{em} &= r_{e2} \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad \vdots \\ k_{em1}d_{e1} + k_{em2}d_{e2} + \dots + k_{emm}d_{em} &= r_{em} \end{aligned} \qquad (2.15)$$

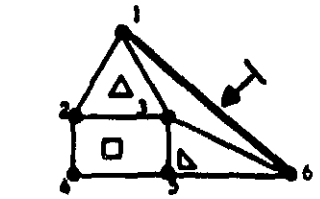
Each DOF in a single element can be numbered from 1 to m, but the DOFs for the entire structure model must be numbered from 1 to n. A simple one-to-one mapping is made from the local element DOF numbering, i.e. 1 to m, to the structure numbering, 1 to n. Equation 2.14 may be written in the same form as (2.15), with the subscripts running from 1 to n. In (2.15), k_{eij} is the elemental stiffness coefficient related to the load on DOF i, and the DOF j. The structure numbering is, of course, different, running from 1 to n, and local DOF i is structure DOF k, and local DOF j is structure DOF l. The elemental stiffness coefficient k_{eij} is thus simply added to the structure stiffness matrix at row k and column l. The mapping is thus determined by the local numbering and the structure numbering.

This procedure is done for all DOFs of all elements in the structure, until all

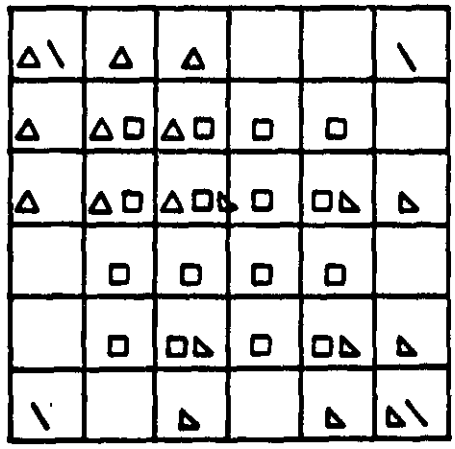
the values of every elemental stiffness matrix have been added into the structure stiffness matrix in their appropriate positions. This is the final structure stiffness matrix [K], and a structure stiffness element k_{kl} is often the addition of several k_{eij}'s. This happens at all the interior nodes of a structure, where more than one element connects and shares a node. For our example of one DOF per node, each connection finite element has a k_{eij} in its elemental stiffness matrix that is mapped to the same k_{kl} in the structure stiffness matrix. In the assembly process outlined above, they are all added to each other as they are assembled into the same location at row k and column l of the structure stiffness matrix. This represents the contribution of each element to the behavior of the mutual (shared) node.

This assembly process easily extends to the more realistic case where more than one DOF is defined at a node. The procedure is the same as above, with the individual DOFs being numbered rather than the nodes. In practice, problems are defined with the nodes numbered. If there are s DOFs per node, the elemental stiffness coefficients corresponding to a node make up an s by s matrix. These matrices are simply assembled at the appropriate locations in the larger structure stiffness matrix by the above rules, by taking into consideration the proper mapping of element (local) DOFs to structure DOFs. Now the structure stiffness matrix is $s \times n$ by $s \times n$, for the case of n nodes and s DOFs per node.

As an example, consider the structure with six nodes in Figure 2-12a. It is made up of a bar element, two dissimilar triangular elements, and a square element, each having one DOF per node. The elements of each elemental stiffness matrix $[K_e]$ are represented by symbols corresponding to that element's shape. Each elemental stiffness matrix value may actually, and usually does, differ from each other, but it is unnecessary to show that to illustrate the process of assembling [K] from the $[K_e]$'s. Since the bar element has two DOFs, its elemental stiffness matrix is two by two, and thus there are four bar symbols in total. Similarly, there are nine symbols for each triangle and 16 for the square element.



(a)



(b)

Figure 2-12: Stiffness Matrix Assembly Example

The structure stiffness matrix [K] is six by six (since there are six nodes and one DOF per node) as shown in Figure 2-12b, where [K] has been assembled by the rules outlined above. Where multiple symbols appear in a stiffness matrix position, their values are added together to form that structure stiffness element. It is easy to see that the pattern of symbol placement within [K] is determined by the structure node numbering. For example, the lower triangle has nodes 3, 5, and 6, and thus its stiffness coefficients appear in rows 3, 5, and 6, at columns 3, 5, and 6. If s DOFs were defined at each node, each symbol would represent an s by s matrix, and the order of [K] would be increased by a factor of s.

The structure equivalent load vector is assembled in the same manner, only in one dimension. As before, in (2.15), the equivalent elemental load r_{ei} is the load associated with DOF i, where i is from the element's local numbering. In the structure numbering, DOF i is k, and thus the load r_{ei} is added to the

structure equivalent load vector in position (row) k. Again, at interior nodes, more than one r_{ei} will add to produce the structure r_k value. The previous remarks hold when s DOFs are defined at a node; in this case, the contribution to the load vector due to one node is an s by 1 vector.

One more topic needs to be covered to complete step 3, and that is rotation transformations. Each finite element is defined and derived for a fixed set of coordinates, and each structure is defined for a fixed set of coordinates. In modelling a structure, some elements may not be oriented properly for the defined structure coordinate system. The elemental stiffness and equivalent load matrices for these elements must undergo a rotation transformation before they are assembled into the structure equations. Figure 2-13 and equations 2.16 and 2.17 show how this is done for a planar rotation in cartesian coordinates.

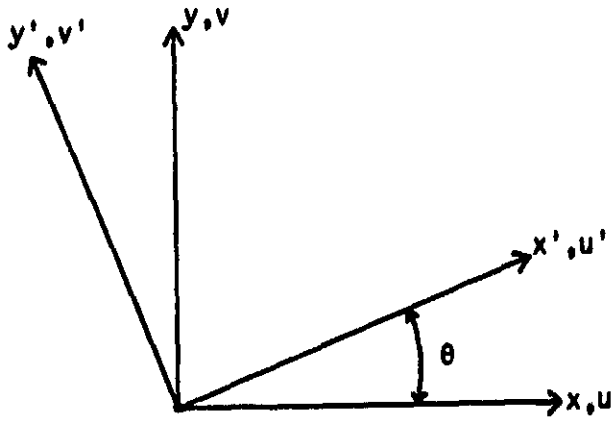


Figure 2-13: Planar Rotation of Cartesian Coordinates

When (x,y) is the structure, or global coordinate system, and (x',y') is the element's, or local coordinate system, the element's equations must undergo a transformation so that they are defined for the (x,y) coordinates, since they are being assembled into structure equations. The relation between the x and y displacements of the two coordinate systems, u and v, define the rotation transformation.

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

or

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = [R] \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.16}$$

The following transformations are performed on the elemental equations.

$$[K_{eg}] = [R]^T[K_e][R]$$

and

$$\{r_{eg}\} = [R]^T\{r_e\} \tag{2.17}$$

where $[K_{eg}]$ and $\{r_{eg}\}$ are the global elemental stiffness matrix and the equivalent load vector used for the structure equation assembly. Hereafter, we omit the "g" subscript on $[K_e]$ and $\{r_e\}$.

Step 4 involves setting up the problem for solution. At this point, (2.14) has been assembled. The $\{r\}$ vector represents the equivalent nodal loads on the DOFs of the structure model, but no external loads have yet been applied at the nodes. The values in $\{r\}$ are only those loads needed to balance any distributed loads, initial stresses, or initial strains on the structure. Finite elements are usually used to analyze structures under various external loading conditions. External loads are defined as those loads applied externally directly to the nodes. They are written as an n by 1 external load vector $\{p_{ext}\}$ (n is the number of DOFs in the model), and incorporated into the finite element problem by adding $\{p_{ext}\}$ to the right-hand side of (2.14). The addition of $\{r\}$ and $\{p_{ext}\}$ forms a new n by 1 vector on the right-hand side, $\{p\}$, simply called the nodal load vector. The result is shown in (2.18), which is the equation for the structure including all loadings. Equation 2.18 is the basic, fundamental finite element problem equation for linear static problems.

$$[K]\{d\} = \{r\} + \{p_{ext}\} = \{p\} \tag{2.18}$$

Equation 2.18 cannot be solved, in its present form, for the unknown DOF vector {d}, since [K] is singular. This occurs because the structure stiffness matrix is a complete description of the relation between all DOFs, and the forces on the DOFs in the structure. That is, the present stiffness matrix represents the structure as if it were floating in space, capable of rigid body movement. A physical interpretation of the stiffness matrix is thus useful. Specifically, element k_{ij} is the force required at DOF i to produce a unit displacement at DOF j, with all other DOFs fixed at zero (assuming all DOFs are displacements). Thus, the rows of the stiffness matrix represent equilibrium forces, and must sum to zero since the sum across a row is a sum over all DOFs (a row of [K] multiplies the DOF vector {d}). Clearly, the stiffness matrix is singular and a unique DOF vector {d} cannot be found for a given loading. An applied load simply produces a rigid body motion; i.e., the entire structure translates and no deformation occurs. Thus, the stiffness matrix must be made non-singular by the application of boundary conditions.

Boundary conditions are usually applied by restraining selected DOFs, forcing their value to be zero or some fixed value. Generally, DOFs are set to be zero, and this is the only type of boundary condition considered in this report. When setting DOFs to be zero, the corresponding row and column of the stiffness matrix, for those DOFs, are deleted. The column is deleted because those values would multiply the zero in the DOF vector {d}, and the row is deleted because the nodal load will include an internal reaction force, which cannot be found until the DOF values are determined. This process results in a non-singular stiffness matrix which will yield a unique solution for {d}. The equations are of the same form as (2.18).

Step 5 is the solution of the system of equations resulting from step 4. Solution of these linear algebraic equations is not trivial because a typical problem may involve thousands of DOFs. This process will be discussed in section 2.7.

The final step in the finite element method algorithm is the determination of stresses and strains from the nodal DOF solutions obtained in step 5. These quantities are solved for directly from the strain-displacement relationship of (2.10), and the stress-strain relationship of (2.5).

2.6 Structural Stiffness Matrix Properties

The structure stiffness matrix resulting from any structural mechanics element formulation has some predictable and useful properties. The stiffness matrix will be positive definite, symmetric, sparse, and banded. There are some problems, however, that result in an indefinite matrix, such as buckling problems, which are not of concern here. One or more of these properties is usually exploited by equation solvers as discussed in section 2.7. It is instructive to briefly examine each of these properties.

The positive definite property (each n by n submatrix, starting in the upper left corner and proceeding down the diagonal of the matrix, has a positive determinant, and hence all diagonal elements of the matrix are positive valued) is evident from any of the variational formulations, specifically the principle of minimum potential energy. To guarantee a minimum rather than a maximum when setting the first variation of the potential energy to zero and solving, the second variation must be positive. If we look at the left-hand side of (2.13), and form another variation with respect to the DOFs {d}, this second variation of the potential energy will be a function of the stiffness coefficients. If one performs all the variational calculus, the stiffness matrix is found to be required to be positive-definite. A more rigorous discussion of this subject may be found in [2].

The symmetry of the stiffness matrix is obvious from considering the equilibrium forces for a DOF represented in each row. The force required at DOF i to produce a unit displacement at DOF j must be the same as the force needed at DOF j to produce a unit displacement at DOF i (all other DOFs fixed to zero). Thus, k_{ij} is equal to k_{ji}. Note that after boundary conditions are

applied, the stiffness matrix remains symmetric. Also, the diagonal elements, k_{ii}, must be positive and non-zero. This is because the force at DOF i needed to produce a unit displacement at the same DOF i must be positive by convention. Thus, the stiffness matrix is usually diagonally dominant (the larger valued elements lie along the diagonal).

The most important property of the stiffness matrix is its sparsity. The reason for this can be seen from (2.12). The elements of the stiffness matrix are derived by integrating over all possible products of the basis functions, which are imbedded in [B] as in (2.10). Since finite element basis functions are defined to go to zero at all other nodes than the one for which they are defined, the product of several basis functions will only be non-zero for nodes on the same element. In other words, the behavior of a given element will only influence adjacent elements. For models with many elements, a single element has no influence on most of the elements in the model, thus the stiffness matrix has many zeros (sparse).

Typically, the percentage of zero elements in the stiffness matrix is 70% to over 90%. Obviously, it is the topology of the structure and the specific finite element model that determines the sparsity. Thus, the percentage of zero elements will be lower in some problems than in others. Elongated structures usually have the most sparse stiffness matrices.

A natural result of the stiffness matrix sparsity is banding of the non-zero elements around the main diagonal. This occurs with proper node numbering of the model. If the model is numbered such that the difference in the node numbers for adjacent nodes is small, tighter banding will occur. The stiffness matrix can have a full band for a poorly numbered model. A rule of thumb for proper numbering is to number across the shorter dimension of the model (in terms of the number of nodes), as illustrated in Figures 2-14 and 2-15. Figure 2-14a is numbered across the short dimension, while Figure 2-14b is numbered

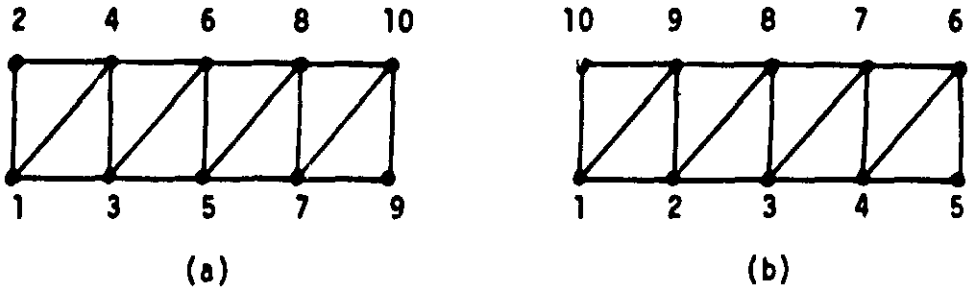


Figure 2-14: Model With Different Node Numbering

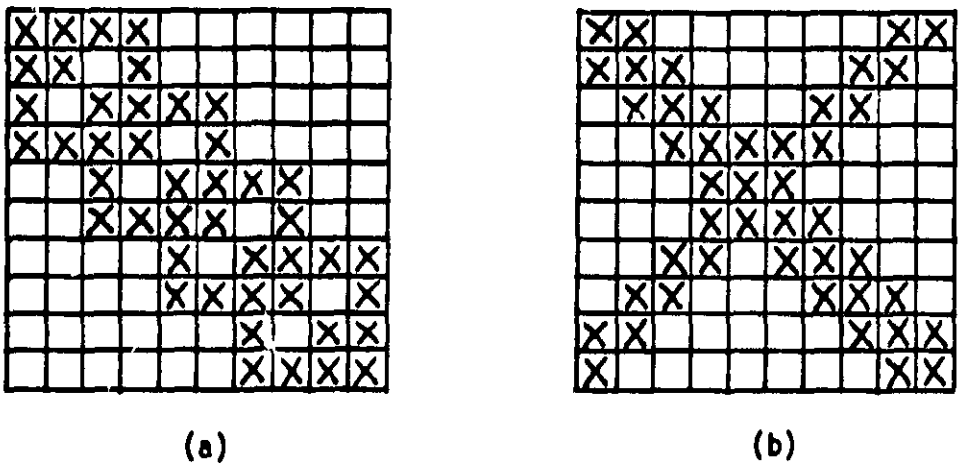


Figure 2-15: Non-zero Stiffness Matrix Entries

across the long dimension. The corresponding non-zero stiffness matrix entries, assuming one DOF per node, are shown by X's in Figure 2-15. The tight band can be observed in Figure 2-15a, while Figure 2-15b shows a full stiffness matrix. Note that the number of non-zero elements is the same in Figures 2-15a and 2-15b.

For a given problem, there is a node numbering scheme that will minimize the bandwidth. This is desireable for some equation solvers, since it can reduce the required storage and number of computations. In general, it is easier to work with a banded matrix, and thus node numbering algorithms are an important part of finite element software packages. There are some cases when such numbering is not desired, as with the frontal solution algorithm discussed in section 2.7.

Given a numbered model, the bandwidth of the structure stiffness matrix [K] can be determined from the equation

$$\text{Bandwidth} = 2ND-1 \tag{2.19}$$

where N is the number of DOFs per node, and D is the maximum difference between the numbers for any two nodes on any element in the model, plus one. The quantity ND is called the semibandwidth of [K]. Equation 2.19 assumes all nodes have the same number of DOFs.

A comment about the dynamic range of the stiffness matrix should be made. The values of the elements in [K] depend on the types of finite elements used, their physical dimensions, and how they are used in modelling the structure. It is common to have a stiffness entry proportional to some length, adjacent to an entry proportional to that length cubed. In a well-formed problem, all element sizes are within one order of magnitude of each other, so that their stiffness matrix coefficient differences are not too severe. However, the nature of the finite element method dictates that [K] will have a large dynamic range (four or five orders of magnitude) for most problems. Subdivision of a model into smaller and smaller elements will reduce the dynamic range, but at a large increase in the cost of problem formulation, solution effort, and complexity. Dynamic range is, of course, extremely problem dependent, precluding any kind of thorough discussion here.

2.7 Solution Methods

After boundary conditions are applied to (2.18), a large set of simultaneous linear algebraic equations results. Solving such a set of equations is always required in the finite element method, where they arise from linear static problems, linearization steps in nonlinear problems, and time-stepping algorithms in dynamic problems. Equation solving is usually the most expensive part of the structural analysis, and may involve from 25% up to 80% of the computations for a problem [1]. For these reasons, the proper choice of an equation solver is important when performing finite element anaylsis.

It should be noted that (2.18) is never solved by inverting the stiffness matrix [K]. A large matrix inversion is costly and time consuming, especially because the inverse of a banded matrix is a full matrix. There is an abundance of literature devoted to sparse matrix and finite element solutions, unfortunately there is yet to be developed an optimal equation solver applicable to every problem. The most efficient solver will depend on the specific problem, and computing hardware available. Thus, only general statements can be made about the merits of available equation solvers.

There are two types of equation solving procedures, direct solvers and indirect (iterative) solvers. Direct solvers are the most popular solution algorithms, since they require a fixed number of numerical operations, given the size and sparsity of the stiffness matrix. Indirect solvers are less attractive, but have some nice features. The number of numerical operations required by indirect solvers is indeterminate, but depends on the conditioning of the matrix equations. We will consider only a direct solution in Chapter 4 of this report.

Direct algorithms usually employ forms of Gaussian elimination, Cholesky decomposition, or the Crout method. Most algorithms factor the stiffness matrix into upper [U] and lower [L] triangular matrices. This step involves the majority of the computational load. The process is outlined in (2.20) below, where the first two lines show the original problem and the factorization. The solution is obtained as indicacted by the last two lines, where solving for {y} is called forward substitution, and solving for {d} is called back substitution. These last two steps are trivial.

$$\begin{aligned} [K]\{d\} &= \{p\} \\ [L][U]\{d\} &= \{p\} \\ [L]\{y\} &= \{p\} \\ [U]\{d\} &= \{y\} \end{aligned} \qquad (2.20)$$

Many options exist for the factorization of [K]. Since [K] is symmetric, it can be factored into the product of a lower triangular matrix [L], and its

transpose $[L]^T$ with Cholesky decomposition. Also, [K] can be factored into the product $[L][D][L]^T$, where [D] is a diagonal matrix. This factorization avoids a square root computation. In Chapter 4 we will use an [L][U] factorization for reasons explained there.

The various algorithms typically exploit the sparsity of the stiffness matrix. A reduction in the required storage and number of computations can be made by operating on just the non-zero band of the stiffness matrix. This is a conventional and popular method. Storage and computational effort can be further reduced by using profile (skyline) storage. In this scheme, the columns of the upper triangular portion of the stiffness matrix are stored beginning with the first non-zero entry in each column which defines the profile, of skyline of the matrix. An index is kept of the location of the diagonal elements in the storage array. With this storage scheme, the model is numbered in order to reduce the infill under the profile, i.e. the number of zeros under the profile. This is because the zeros may be changed to non-zero numbers during factorization. The triangular factorization may be done efficiently with dot product routines. A more in depth discussion may be found in [6].

A popular direct method which conserves storage is the wavefront or frontal solution method [7]. This method uses Gauss elimination, and stiffness coefficients related to a node are reduced as soon as all equations contributing to that node have been assembled. Thus, as each finite element is assembled into [K], Gauss reduction begins on those equations which are complete. This method relies on numbering elements rather than nodes. The advantage to a frontal solver is that storage requirements are reduced, but at the cost of complex programming and equation manipulation overhead.

Indirect methods use iterative procedures that require an indefinite number of operations to converge to an acceptable solution. One iterative solver, indicative of the others, is Gauss-Seidel iteration. This algorithm refines an initial estimate in

a procedure using an over-relaxation factor ω. The algorithm is guaranteed to converge for positive-definite, symmetric systems, if ω is between zero and two. The basic algorithm is outlined below [1], for solution of the N^{th} order system of equations $[K]\{d\}=\{p\}$:

$$d_i^{n+1} = d_i^n + (\omega/k_{ii})(p_i - \sum_{j=1}^{i-1} k_{ij} d_j^{n+1} - \sum_{j=1}^{N} k_{ij} d_j^n) \qquad (2.21)$$

where n is the iteration number, and the process begins with an initial estimate $\{d_0\}$ for $\{d\}$.

Indirect (iterative) methods have a few advantages: they are easier to program than direct methods, they demand less storage, and fast computation times are obtainable for low-accuracy solutions, or for good initial estimates of $\{d\}$. Direct algorithms have other advantages when a given problem has multiple loading conditions, i.e. multiple $\{p\}$ vectors. In this case, the factorization only needs to be computed once to solve for all of the loadings. Multiple $\{p\}$ vectors often occur in finite element problems. With an indirect solver, the entire algorithm must be used for every $\{p\}$ vector, and the number of computations for each solution is indeterminate. In general, direct algorithms require less numerical operations and are the preferred algorithms. For these reasons, a direct algorithm is initially investigated in this paper.

2.8 Nonlinear and Dynamic Problems

The finite element method has been described thus far only for linear static problems. However, many important and practical problems require modelling of nonlinear behavior and/or time-dependent (dynamic) loads. The equations for these problems are derived with the same methods used for linear, static problems, but their solutions can no longer be obtained by solving a single matrix equation such as (2.18).

In nonlinear problems, the stiffness matrix is a function of displacements, and

will be changing throughout the structure deformation. In dynamic problems, the loads will change with time, and the displacements will also be time-varying. Solutions for these problems require much more computational effort than the linear static problems, and several orders of magnitude more work for a nonlinear dynamic problem. A brief discussion of the problem formulations and solution methods will be given here. A more thorough account of the subject may be found in [3].

2.8.1 Nonlinear Problems

In the linear problems, the differential equations governing the behavior of the structure were linear This was due to two inherent assumptions for elastic structures [3]:

1. Linear strain-displacement relationships.
2. Linear stress-strain relationships.

Two types of nonlinearities may be defined. If the first assumption is not valid, the problem posesses geometric nonlinearity. This means that deformations are not small, and that the structure geometry changes significantly during loading. If the second assumption is not valid, material nonlinearities are present, which implies that material properties change under loading. Geometric nonlinearities are usually more severe, but both types may be combined into problem formulations that may be solved by the following methods.

The basic equation (such as (2.18)) for a nonlinear static problem may be written as follows, where [K] is a function of the displacements,

$$[K(d)]\{d\} - \{p\} = 0$$

or

$$[K(d)]\{d\} = \{p\} \tag{2.22}$$

Four popular numerical procedures will be described for solving (2.22) [3].

The first method is direct iteration. This is the most straightforward and

least sophisticated method. An initial estimate, $\{d\}=\{d_0\}$ is assumed for the displacements of (2.22). An improvement of the estimate is found by

$$\{d_1\} = [K(d_0)]^{-1}\{p\} \tag{2.23}$$

or, in general,

$$\{d_i\} = [K(d_{i-1})]^{-1}\{p\} \tag{2.24}$$

where i is the iteration number. Of course, the matrix inversion is just notational and never performed, rather direct equation solving techniques are used. The method terminates when two successive approximations are within a defined tolerance. One possible convergence criteria is defined by

$$\{e_i\} = \{d_i\} - \{d_{i-1}\} \tag{2.25}$$

and convergence is achieved when

$$\|\{e_i\}\| < \alpha\|\{d_i\}\| \tag{2.26}$$

where $\|\cdot\|$ is some norm and α is some fraction.

The direct iteration method is very simple, but may be divergent if certain nonlinearities are modelled. At each iteration, a different equation of the form of (2.24) must be solved. Thus n times more work than a linear problem is needed in the solution process, where n is the number of iterations.

The second method is Newton-Raphson iteration. This method requires linearization of (2.22) with a truncated Taylor series expansion, which results in the equation

$$[K(d_{i-1})]\{d_{i-1}\} - \{p\} + [K_{i-1}^t](\{d_i\}-\{d_{i-1}\}) = 0 \tag{2.27}$$

The matrix $[K^t]$ is the tangential stiffness matrix, which is the derivative of $[K(d)]$ with respect to the displacements. An initial estimate, $\{d\}=\{d_0\}$ is used, and refined by (2.27) in the form

$$[K_{i-1}^t]\{\Delta d_i\} = \{p\} - [K(d_{i-1})]\{d_{i-1}\} \tag{2.28}$$

where

$$\{\Delta d_i\} = \{d_i\} - \{d_{i-1}\} \tag{2.29}$$

The same convergence criteria for the direct iteration method can be used for this method.

The Newton-Raphson method converges faster than the direct iteration method given the same $\{d_0\}$, and, in fact, it converges quadratically near the solution. The initial estimate, however, must be in the problem's region of convergence, or the process will diverge. This numerical technique is very standard, and information and examples may be found in any good numerical methods text, and in [8] and [10]. Note at each iteration the linearization, i.e. the evaluation of the tangential stiffness matrix, must be performed along with solving (2.28).

The third method is a modified Newton-Raphson method. At some point in the standard Newton-Raphson procedure, it may become economical to use the most recent linearization for all, or some, subsequent iterations. This provides a trade-off between computation time in the evaluation of $[K^t]$ at each step, and the rate of convergence, which may prove advantageous. The equation to be solved at each step is

$$[K_c^t]\{\Delta d_i\} = \{p\} - [K(d_{i-1})]\{d_{i-1}\} \tag{2.30}$$

where $[K_c^t]$ is a tangential stiffness matrix that is used for several iterations. Again, information on this modified algorithm can be found in most numerical methods texts [10], and an example is given in [8].

The last method we will present is the incremental load method, which solves the incremental equation

$$[K_i^t]\{\Delta d_i\} = \{\Delta p_i\} \tag{2.31}$$

where

$$\{\Delta d_i\} = \{d_i\} - \{d_{i-1}\}$$

and

$$\{\Delta p_i\} = \{p_i\} - \{p_{i-1}\} \tag{2.32}$$

Here again, the tangential stiffness matrix is required, and incremental displacements are solved for as the load is incremented. For almost all structures, the initial displacement and load vectors can be the zero vector. The tangential stiffness matrix $[K^t]$ and the incremental nodal load vector $\{\Delta p_i\}$ are evaluated, and (2.31) is solved for the incremental displacement vector $\{\Delta d_i\}$. The current state of $\{d\}$ is then calculated as

$$\{d_i\} = \{d_{i-1}\} + \{\Delta d_i\} \tag{2.33}$$

At this point, the iterations could continue by evaluating $[K^t]$ from the $\{d_i\}$ from (2.33), applying the next load increment, i.e. using (2.32) to form $\{\Delta p_{i+1}\}$, and solving (2.31) for $\{\Delta d_{i+1}\}$. However, since $[K^t]$ is a linear approximation to the nonlinear curve, the $\{p_i\}$ vector does not exactly satisfy (2.22) with the $\{d_i\}$ vector.

An equilibrium load correction step can be used to keep the linear approximations closer to the actual nonlinear curve. After solving (2.31) then (2.32) in an iteration, the next step is to find the actual $\{p\}$ vector that satisfies (2.22) by performing that matrix-vector operation

$$[K(d_i)]\{d_i\} = \{p_i^*\} \tag{2.34}$$

and obtaining $\{p_i^*\}$. In the next iteration, $\{p_{i-1}\}=\{p_i^*\}$ instead of $\{p_i\}$ used in (2.31), when applying the load increment. Thus, the load is incremented from the exact load value on the nonlinear curve satisfying (2.22) with the last displacement vector. This procedure is a more accurate approximation than when the correction step is omitted. An example of this algorithm may be found in [8].

The incremental load method will converge for any problem if the load increments are taken small enough. However, if $[K^t]$ is exactly the zero matrix at any iteration, the algorithm may diverge, and thus adjustments must be made to

avoid this. At each iteration of the algorithm, the tangential stiffness matrix must be evaluated, (2.31) must be solved, and the matrix-vector product of (2.34) must be performed.

The nonlinear solution methods presented here are by no means the only ones available, and each method has variations and improvements. The best method to use is purely problem dependent, and often a series of different methods will be used when solving a nonlinear problem. If one has a good initial estimate of the solution, the Newton-Raphson method may be the best choice. As a general solution method, the incremental load algorithm is attractive because it always converges.

2.8.2 Dynamic Problems

Dynamic problems need to be solved when the applied loads are time-varying. Analytic solution techniques may be used for some problems, but they do not easily accomodate nonlinear problems or solution of transients. Discretization of the time dimension leads to step-by-step formulations which provide a better treatment of transients, and allow for nonlinear analysis. Solution of dynamic problems is a large and complicated subject. Thus, only a few solution methods will be discussed here. A more thorough presentation may be found in [3].

Analytic solution procedures are applied on a set of dynamic finite element equations that have, of course, been discretized in space, but not in time. In general, an ordinary differential matrix equation describing a time-varying finite element problem can be written as

$$[M]\{\ddot{d}\} + [C]\{\dot{d}\} + [K]\{d\} = \{p(t)\} \qquad (2.35)$$

Standard analytical techniques for solving differential equations may be applied to (2.35), with some added approximations.

In (2.35), the quantities d, $\dot{d}$, and $\ddot{d}$ are the nodal displacements, velocities, and accelerations, respectively. The matrix [M] is the mass matrix, which

describes the distribution of mass throughout the structure. It may be formulated as a lumped (diagonal) mass matrix, or a consistent (same band structure as [K]) mass matrix. The matrix [C] is a damping matrix which usually has the same band structure as [K], although lumped formulations exist. The damping matrix is difficult to determine, so it is usually defined as a linear combination of [M] and [K]

$$[C] = a[M] + b[K] \tag{2.36}$$

where a and b are usually determined experimentally. The matrix [K] is the usual stiffness matrix, and the vector {p(t)} contains the nodal time-varying loads.

We will describe two anayltic solution techniques for a specific and a general case of (2.35). First, consider the free vibration problem where the damping and load vector are zero, resulting in the equation

$$[M]\{d\} + [K]\{d\} = 0 \tag{2.37}$$

Assuming a solution of the form $\{d\}=\{d'\}\sin\omega t$ and substituting it into (2.37) gives

$$-\omega^2[M]\{d'\} + [K]\{d'\} = 0$$

or

$$(-\omega^2[M] + [K])\{d'\} = 0 \tag{2.38}$$

and the characteristic equation yields the following eigenvalue problem,

$$[K]\{d'\} = \omega^2[M]\{d'\} \tag{2.39}$$

Equation 2.39 may be solved by a variety of standard methods. If [M] and [K] have dimension n by n, then n values (eigenvalues) of ω^2 can be found, which represent the natural frequencies of the system. They will be real-valued if [M] and [K] are positive definite, which is usually the case. The n corresponding eigenvectors {d'} represent the natural modes of the system. This free vibration problem is just one simple example of how basic differential equation solution techniques may be applied to (2.35).

The analytical procedure of modal decomposition analysis may be used to solve (2.35) and provide the transient response [3]. The general solution for the free response of (2.35) can be shown to be

$$\{d\} = \{d'\}e^{\alpha t} = \sum_{i=1}^{n}\{d_i'\}e^{\alpha_i t} \tag{2.40}$$

where α_i and $\{d'_i\}$ are the eigenvalue and eigenvector for mode i. We will assume that the forced response ($\{p(t)\}$ nonzero) may be written as a linear combination of the modes

$$\{d\} = \sum\{d_i'\}y_i = [\{d_1'\}, \{d_2'\}, \ldots, \{d_n'\}]\{y\} \tag{2.41}$$

where y_i is the scalar mode participation factor, and is a function of time, $y_i=y_i(t)$. Substituting (2.41) into (2.35), and premultiplication of each mode equation by $\{d_i\}^T$, i=1 to n, we obtain the set of scalar, independent equations

$$m_i\ddot{y}_i + c_i\dot{y}_i + k_i y_i = p_i \tag{2.42}$$

where

$$\begin{aligned} m_i &= \{d_i'\}^T[M][d_i'] \\ c_i &= \{d_i'\}^T[C][d_i'] \\ k_i &= \{d_i'\}^T[K][d_i'] \\ p_i &= \{d_i'\}^T\{p\} \end{aligned} \tag{2.43}$$

These n scalar equations in (2.42) may be solved independently, and the total response may be found by superposition according to (2.41).

Solving the general free response eigenvalue problem of (2.35) is difficult because the eigenvalues and eigenvectors are, in general, complex-valued. In practice, the real eigenvalues found by solving (2.39) are used. Decoupled equations, i.e. (2.42), and real y_i values will still occur if the [C] matrix is formed according to (2.36). The entire eigenvalue problem does not need to be solved,

as only a few of the low frequency modes need to be considered. This is because the high frequency response is usually critically damped, and does not contribute much to the total response.

A step-by-step or recurrence relation may be formulated by discretization of the time dimension. These methods incorporate initial conditions, and thus provide transient analysis. The discretized finite element equation, in space and time, may be written as [8],

$$[M]\{\Delta d_i\} + [C]\{\Delta d_i\} + [K]\{\Delta d_i\} = \{\Delta p_i\} \tag{2.44}$$

where

$$\{\Delta d_i\} = \{d_i\} - \{d_{i-1}\}$$
$$\{\Delta d_i\} = \{d_i\} - \{d_{i-1}\}$$
$$\{\Delta d_i\} = \{d_i\} - \{d_{i-1}\}$$

and

$$\{\Delta p_i\} = \{p_i\} - \{p_{i-1}\}$$

and i is the time step.

Many popular recurrence methods have been derived by Newmark [9], Wilson [11], Houbolts [12], and Hilber [13]. A good example of these types of derivations that include the basis functions in the time dimension may be found in [3]. One such algorithm may be written as [8],

$$(a_1[M] + b_1[C] + c_1[K])\{\Delta d_i\} = \{\Delta p_i^*\} \tag{2.45}$$

where

$$\{\Delta p_i^*\} = \{\Delta p_i\} + [M](a_2\{d_{i-1}\}+a_3\{d_{i-1}\})$$
$$+ [C](b_2\{d_{i-1}\}+b_3\{d_{i-1}\}) + c_2[K]\{\Delta d_{i-1}\} \tag{2.46}$$

For the method proposed by Newmark, the constants are given by

$$a_1 = 1/(\beta\Delta t^2) \quad b_1 = \alpha/(\beta\Delta t) \quad c_1 = 1$$

$$a_2 = a_1\Delta t \qquad b_2 = b_1\Delta t \qquad c_2 = 0 \tag{2.47}$$
$$a_3 = 1/(2\beta) \qquad b_3 = [(\alpha/2\beta) - 1]\Delta t$$

The quantity Δt is the length of each time step, and the constants α and β affect the behavior and stability of the algorithm. Discussions on good choices of α and β may be found in [3], along with many other references. In this case, a good choice is α=0.5 and β=0.5. Initial conditions governing the problem are introduced to begin the step-by-step calculation.

Note that the solution of (2.45), or one time step, involves as much computational effort (once the equations are formulated) as a linear static problem. If m time steps are used, then the solution of a dynamic problem with a recurrence algorithm is m times as costly as a static problem.

The conventional analytic solution procedures are obviously not applicable to nonlinear dynamic problems, because the nonlinear equations change throughout the loading process. However, the step-by-step algorithms are quite useful. Nonlinear dynamic finite element problems require the most computation. Other solution methods besides those noted here are available and are sometimes more efficient for certain problems.

The recurrence algorithms mentioned earlier may be used to solve nonlinear dynamic problems, but with the additional complexity that one or more of the matrices [M], [C], [K] are dependent on {d} and change every time step. The step-by-step relation of (2.45) is quite applicable. An equilibrium load correction for each time (load) step can be included in the algorithm by using the following expression for the incremental load in (2.46), as defined for Newmark's method [8],

$$\{\Delta p_i\} = \{p_i\} - ([M]\{d_{i-1}\} + [C]\{d_{i-1}\} + \{p_{i-1}\}) \tag{2.48}$$

It is often necessary to iterate within each time step to reduce the residual

error and track the nonlinear curve more accurately. Any of the iterative nonlinear solution schemes mentioned earlier may be used. Instead of iterations within time steps, an extrapolation of previous [M], [C], and [K] values may be used. These methods are briefly described and referenced in [3].

Finally, it should be stated that nonlinear and dynamic finite element problems always present a more difficult equation solving task than linear static problems, especially when a problem is both nonlinear and dynamic. In most cases, however, the equations to be solved are a system of linear algebraic equations, much like (2.18) for the linear static problems. Thus, the computations are alike, but many more of them must be performed. The most obvious exception to this is the use of analytic procedures for linear dynamic problems, where eigenvalue problems need to be solved.

2.9 Summary and Conclusion

This chapter has presented an outline and explanation of the basic finite element method for structural mechanics problems. The fundamentals of the method were described, followed by an example of the finite element equation formulation for a plane strain triangular element. Element and stiffness matrix assembly were described, and the properties of the stiffness matrix were detailed. Emphasis was placed on linear static problems throughout the chapter, and their solution methods were discussed in section 2.7. In section 2.8, solution methods for nonlinear and dynamic problems were covered.

This introduction to finite elements was by no means comprehensive, but it provides the necessary background to understand the nature of structural mechanics problems, and what type of equation solution is required. Essentially, all linear static, and most nonlinear and dynamic problems, can be broken down into one or many equations of the form of (2.18), where [K] always has the properties described in section 2.6. An example of a linear static finite element problem, using plate bending elements, is described in Chapter 3.

3. Finite Element Case Study

3.1 Introduction

This chapter describes a specific finite element case study. It will be used to quantify the requirements of a finite element processor. In forthcoming research, the case study will be used to evaluate the performance of our finite element optical processor. The problem formulated concerns the bending of plates, which is a common application area in structural mechanics. The finite element formulation, and assembly of the structure stiffness matrix [K] is described. The stiffness matrix properties, discussed in section 2.6 are quantified for this model.

3.2 Case Study Structure

A simple structure is needed for our initial application of finite element structural analysis. The problem of plate bending was chosen because it is an important, frequently used, and well-defined type of finite element analysis. The structure chosen, very modestly and quite arbitrarily, was a 6 foot by 8 foot by 1 inch thick plate of aluminum. Since the larger two dimensions are close to a square, and since the thickness is much less than either the length or the width, good plate bending results should be obtainable with finite element analysis of this structure.

The material for the plate was chosen to be T4-2024 aluminum. Its structural properties are listed below:

$E = 10.6 \times 10^6$ lbs/in.^{2}	MODULUS OF ELASTICITY
$G = 4.00 \times 10^6$ lbs/in.^{2}	SHEAR MODULUS
$\nu = 0.325$	POISSON'S RATIO

The plate and its reference coordinate axes are shown in Figure 3-1. We will consider the plate to be made of an isotropic material.

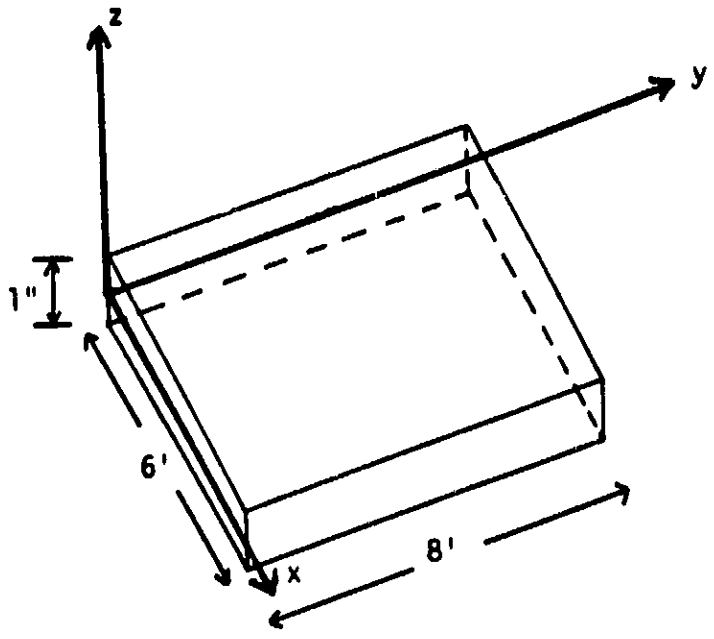


Figure 3-1: Case Study Aluminum Plate

There are some pertinent applications for modelling such a structure. Structural analysis of the deck of a ship, or the floor of a building may require plate bending anaylsis of similar structures. In problems such as these, a designer is often interested only in what happens to the structure under large loads, so that they can be designed to the proper safety factor. In these cases, a very precise finite element solution is not required, since the user is not concerned with many decimal places of accuracy. However, in other applications such as analysis of part of an airframe or a satellite, a designer may be concerned about very small displacements because of restrictive tolerances in the structure, or even the dynamic response. In these cases a very accurate analysis is needed. Thus, the requirements of a finite element analysis can differ greatly from application to application.

3.3 Plate Bending Finite Element Case Study Derivation

The finite element which will be used to discretize and model the structure of Figure 3-1 is derived in Chapter 10 of [3]. Important aspects of the derivation will be outlined here. We have made a change to the derivation and the subsequent elemental stiffness matrix $[K_e]$ presented in [3]. We use a right-handed coordinate system rather than a left-handed one as in [3]. This change is made by a simple adjustment of the elemental stiffness matrix equation, as will be pointed out later in this section.

The concepts for this derivation arise from thin plate theory, where the plate thickness, or the z dimension, is small compared to the size of the plate. In plate bending analysis, as defined for Figure 3-1, only the displacement of the plate in the z direction is defined and is of concern. The displacement for any point on a plate may be written as a displacement field, w(x,y), in terms of the x and y coordinates. Expressions for plate bending stresses, strains, and the resultant forces, and other structural mechanics plate bending equations may be found in [1], [3], [14], and other texts. Full details cannot easily be included here, and they are not vital for presentation of the concepts and attributes of a plate bending finite element problem.

A rectangular plate bending finite element can be defined as in [3], as shown in Figure 3-2. The sides of each element are of lengths 2a and 2b, and the element thickness is t. The element has four nodes, one at each corner of the rectangle. Although it is usually of little consequence since the plates are thin, it should be noted that all plates and plate bending elements are oriented with the x-y plane passing through the centroid, or the middle of the plate (since it is uniform) in z, as shown in Figure 3-1.

To provide a reasonable amount of continuity across elements, three DOFs are defined at each node n : the displacement w_n in the z direction, a rotation

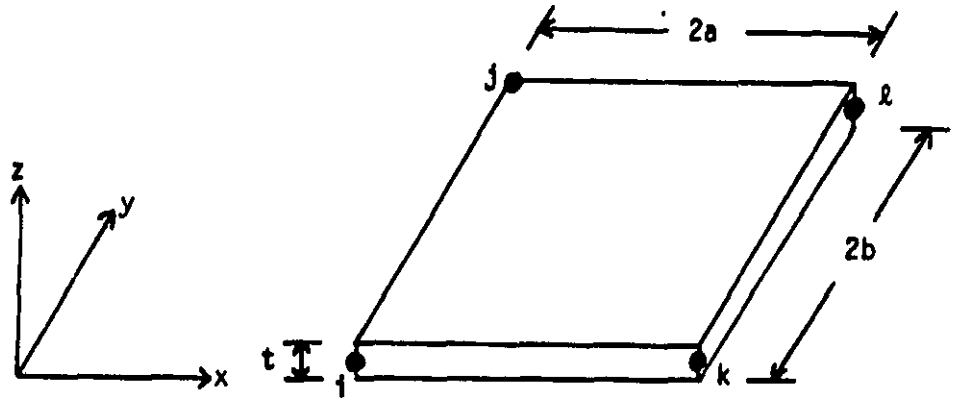


Figure 3-2: Rectangular Plate Bending Element

$\theta_{xn}=\partial w_n/\partial x$ about the x-axis, and a rotation $\theta_{yn}=-\partial w_n/\partial y$ about the y-axis. The loads corresponding to these nodal parameters are a force in the z direction, a moment, or couple, about the x-axis, and a moment about the y-axis. The nodal DOFs and loads are shown in Figures 3-3a and 3-3b, respectively. The directions of the rotations are determined by the right-hand rule.

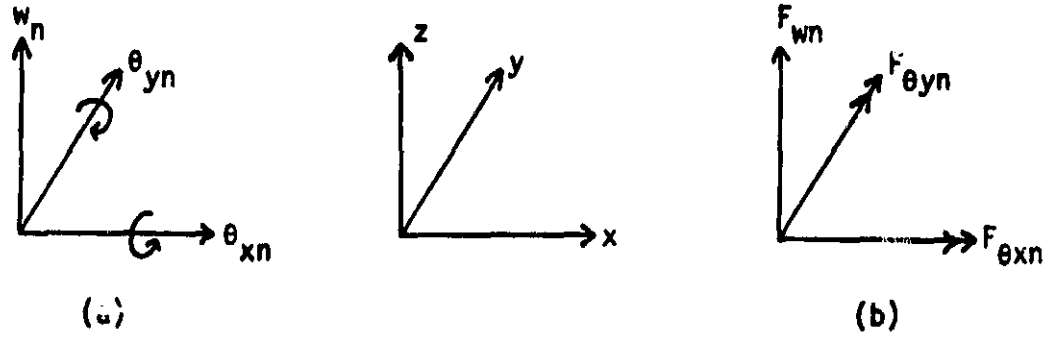


Figure 3-3: Nodal DOFs and Loads for Case Study

It immediately follows that the DOF and load vectors for one node n may be written as

$$\{\delta_n\} = \begin{bmatrix} w_n \\ \theta_{xn} \\ \theta_{yn} \end{bmatrix}$$

and

$$\{r_n\} = \begin{bmatrix} F_{wn} \\ F_{\theta xn} \\ F_{\theta yn} \end{bmatrix} \quad (3.1)$$

The element DOF and load vectors for one element e consist of the vectors in (3.1) and are defined as

$$\{d_e\} = \begin{bmatrix} \delta_i \\ \delta_j \\ \delta_k \\ \delta_l \end{bmatrix}$$

and

$$\{r_e\} = \begin{bmatrix} r_i \\ r_j \\ r_k \\ r_l \end{bmatrix} \quad (3.2)$$

where the subscripts i-l denote the four nodes on an element (see Figure 3-2). The order of the nodal DOFs and loads within all element vectors is extremely important and must be kept the same (i.e. the notation in Figure 3-2 for all nodes). The node lettering in Figure 3-2 is just one ordering defined for this problem in [3]. Others could be used, but the entire problem formulation must stay consistent with those. We note that the vectors in (3.2) are those in (2.13),

$$[K_e]\{d_e\} = \{r_e\} \quad (3.3)$$

We next consider obtaining an expression for the elemental stiffness matrix $[K_e]$. An equation formulation similar to the one described in section 2.4 must be performed, and an equation of the form of (2.12) must be evaluated for $[K_e]$. The $\{r_e\}$ equation is not evaluated for this case study because no distributed loads, or initial stresses or strains are involved. The basis functions for each DOF involve quite complicated expressions and are given in [3], along with the other significant formulation steps that will not be repeated here. The proper equations are evaluated and an explicit equation for the elemental stiffness matrix is given as [3],

$$[K_e] = (1/(60ab))[L]\langle D_x[K_1]+D_y[K_2]+D_1[K_3]+D_{xy}[K_4]\rangle[L] \qquad (3.4)$$

where

$$D_x = D_y = Et^3/[12(1-\nu^2)]$$

$$D_1 = \nu D_x$$

and

$$D_{xy} = [(1-\nu)/2]D_x \qquad (3.5)$$

are elements of the elasticity matrix and

$$[L] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.6)$$

and

$$[I] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2a & 0 \\ 0 & 0 & 2b \end{bmatrix} \qquad (3.7)$$

which depends on the size of the finite element. For four nodes per element and three DOFs per node, the matrices $[K_1]$ - $[K_4]$ are 12 by 12 matrices given in [3]. From equations 3.4 through 3.7, the contributions of material properties (E,ν) and element sizes (a,b,t) are apparent. Thus, (3.3) is defined in terms of the vectors in (3.2) and the matrix in (3.4).

As mentioned earlier, the finite element formulation was performed in [3] for a left-handed coordinate system. It is a simple matter to convert (3.4) to a right-handed expression. This is the representation we use in this study. To achieve this, only (3.7) needs to be revised for the right-handed system in Figure 3-2 as :

$$[I] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2a & 0 \\ 0 & 0 & -2b \end{bmatrix} \qquad (3.8)$$

The displacement field w(x,y) for an element is determined by the twelve nodal parameters, the $\{d_e\}$ vector. It may be written as a polynomial expression with 12 terms as

$$w(x,y) = \alpha_1+\alpha_2x+\alpha_3y+\alpha_4x^2+\alpha_5xy +\alpha_6y^2+\alpha_7x^3+\alpha_8x^2y+\alpha_9xy^2+\alpha_{10}y^3+\alpha_{11}x^3y+\alpha_{12}xy^3 \quad (3.9)$$

Equation 3.9 includes all terms through third order and two fourth order terms. The α's represent 12 unknown constants which are determined by the 12 nodal DOFs and their basis functions.

We now discuss continuity of the plate bending element. The displacement field (3.9) varies as a cubic in x for a constant y value, and as a cubic in y for a constant x value. Thus, along the element boundaries, which are the interfaces between adjacent elements, w(x,y) will vary as a cubic polynomial. To examine the continuity across elements, we consider the edge i-j in Figure 3-2. Since this edge lies on a line of constant x, the following equations may be written for w(x,y) on this edge (in terms of new coefficients c_i) as

$$w(x,y) = c_1 + c_2y + c_3y^2 + c_4y^3 \quad (3.10)$$

$$\partial w(x,y)/\partial y = c_2 + 2c_3y + 3c_4y^2 \quad (3.11)$$

$$\partial w(x,y)/\partial x = c_5 + c_6y + c_7y^2 + c_8y^3 \quad (3.12)$$

where (3.11) is obtained by taking the derivative of (3.9) with respect to y, and then substituting x=constant. Equation (3.12) is obtained similarly.

The four nodal DOFs w_i, θ_{yi}, w_j, θ_{yj}, completely specify the four c_i and thus the variations described by (3.10) and (3.11). Thus, the finite element approximations for w(x,y) and its tangential derivative will be continuous across elements. However, the normal derivative (3.12) along the edge is a cubic and only two DOFs, θ_{xi}, θ_{xj}, remain to describe (3.12). Since a cubic cannot be uniquely specified by two values, normal derivatives at element interfaces will not

be continuous. However, it has been well established and proven theoretically that this plate bending element is convergent.

3.4 Case Study Discretization and Stiffness Matrix Assembly

The plate bending finite element case study problem and model defined in section 3.3 is used to discretize the structure described in section 3.2, and shown in Figure 3-1. The aluminum plate is divided into eight elements, each 1 inch thick and of size 3 feet in the x dimension and 2 feet in the y dimension. The dimensions of each element (Figure 3-2) are: a=18 inches, b=12 inches, t=1 inch.

The discretized structure to be used is shown in Figure 3-4. This model has 8 elements, 15 nodes, and 15x3=45 DOFs (for our case of 3 DOFs per node). Each element is oriented in the structure as defined in Figure 3-2. No rotation transformations are needed. Each element in the model is identical, and hence evaluation of (3.4) is required only once to assemble [K] for the entire model.

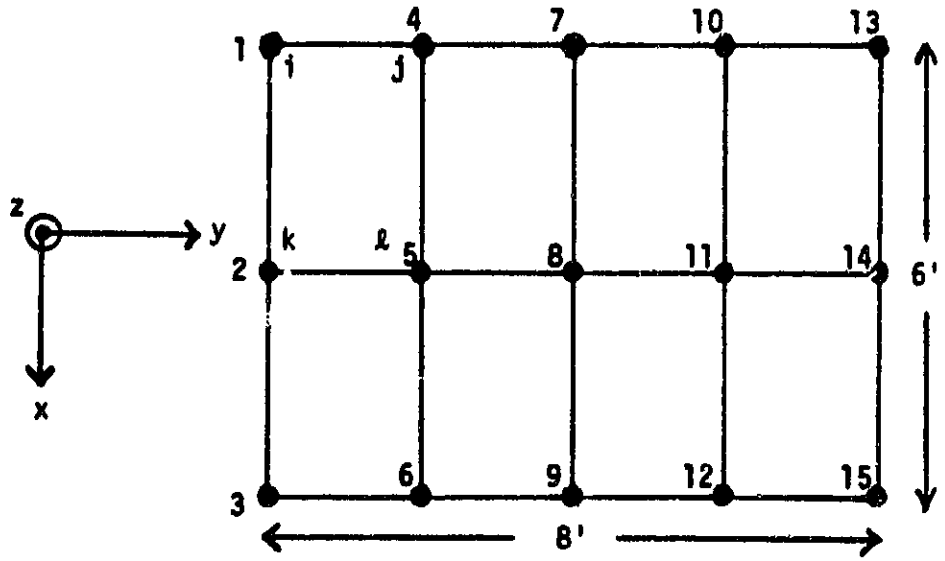


Figure 3-4: Discretized Structure

The nodes in the model of Figure 3-4 are numbered to achieve minimum bandwidth for [K] as outlined in section 2.6. Specifically, we number nodes across the dimension of the model with the least number of nodes, the x dimension. The bandwidth is given by (2.19), with N=3 DOFs per node and D=5, to be BW=2(3)(5)-1=29. The structure stiffness matrix is 45 by 45 since there are 45 DOFs in the model. A matrix bandwidth of even 29 still indicates a significant matrix sparsity.

For our case study, equation 3.4 was evaluated digitally. The computed 12 by 12 elemental stiffness matrix used in our simulations is given in Appendix I. The structure stiffness matrix is assembled according to the rules given in section 2.5. For our case study, the assembly process consists of adding the elements of eight identical elemental stiffness matrices into the proper locations in the 45 by 45 structure stiffness matrix.

It would be extremely cumbersome to detail the entire assembly process here, although stiffness matrix assembly, node numbering, and other problem formulation tasks are significant parts of a finite element problem. This report will concentrate on the solution of such problems, assuming they are formulated as detailed in section 2.4. However, we will detail how one of the eight elements is assembled, to illustrate the process described in section 2.5.

Consider the element of Figure 3-4 with nodes (5,8,6,9), corresponding to the (i,j,k,l) ordering defined in Figure 3-2 and equation 3.2. Realize that the nodes of any single element will be numbered locally (1,2,3,4) corresponding to the (i,j,k,l) in Figure 3-2, as shown in Figure 3-5. A local element is a structure element with the nodes renumbered beginning with 1. Thus the mapping of local node numbering to structure node numbering for the element (5,8,6,9) is shown in Table 3.1.

Let the 12 DOFs (3 per node for the 4 nodes of this element) as described by (3.2) be numbered 1-12. Let the 45 structure DOFs, as described by (3.13),

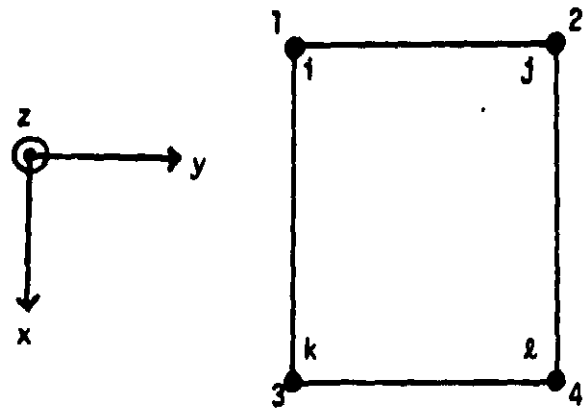


Figure 3-5: Local Element Numbering

Local Element Numbering	Structure Element Numbering
1	5
2	8
3	6
4	9

Table 3-1: Local to Structure Node Number Mapping

$$\{d\} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \cdot \\ \cdot \\ \cdot \\ \delta_{15} \end{bmatrix} \qquad (3.13)$$

be numbered 1-45. Local node 1 has local DOFs 1,2,3 . Local node 1 is structure node 5, which has structure DOFs 13,14,15 corresponding to δ_5 in (3.13). This local to structure DOF mapping is continued for the other three nodes with the results shown in Table 3.2.

The mapping in Table 3.2 completely defines the assembly process. Every element k_{eij} in the elemental stiffness matrix of element (5,8,6,9), where i and j

Local Element DOFs	Structure Element DOFs
1,2,3	13,14,15
4,5,6	22,23,24
7,8,9	16,17,18
10,11,12	25,26,27

Table 3-2: Local to Structure DOF Mapping

are the local element DOF numbers, simply is added to position k_{mn} in the structure stiffness matrix, where m and n are the structure element DOF numbers corresponding to i and j, according to the mapping in Table 3.2. In other words, the 144 elemental stiffness matrix elements are added to 144 locations of the structure stiffness matrix (which has 45×45=2025 elements) by the mapping in Table 3.2. For further insight, some specific assembly examples are given in Table 3.3.

Elemental Stiffness Matrix Element	is added to	Structure Stiffness Matrix Position
(1,1)		(13,13)
(4,9)		(22,18)
(10,11)		(25,26)
(2,8)		(14,17)

Table 3-3: Elemental Stiffness Matrix Assembly Examples for Element (5,8,6,9) of Figure3-3

The result of this large and tedious assembly procedure is the 45 by 45 structure stiffness matrix [K], with a non-zero element bandwidth of 29. The stiffness matrix possesses all the properties discussed in section 2.6, some of which will be detailed in the next section.

3.5 Case Study Stiffness Matrix Details

The completely assembled structure stiffness matrix [K] is listed in Appendix II. A factor of 10^6 has been factored out of the matrix. Pounds are the units for the loads, inches for the displacements, and radians for the rotations. It can readily be seen that the stiffness matrix is symmetric and diagonally dominant. All the diagonal entries are positive (a necessary condition for a positive definite matrix).

The sparsity of the stiffness matrix is rather pronounced, even for such a small case study problem. There are 1140 entries in the band, out of a total of 2025 stiffness matrix elements. The bandedness of the problem guarantees that at least 43.7% of the elements in [K] are zero. However, about half the band elements are also zero, yielding 557 or 28.5% non-zero elements in [K]. This matrix is a good example of how a profile storage technique can take advantage of the zeros within the band for a more efficient storage scheme than band storage. However, because of the parallel nature of our processor, such techniques are not useful in this study.

The dynamic range of the stiffness matrix was found to be quite large, as expected. The smallest non-zero magnitude is 0.00021152; the largest non-zero magnitude is 8.37760067. Thus, the dynamic range is almost five orders of magnitude. An accurate solution of this finite element problem requires a processor which can adequately represent the entire dynamic range of the stiffness matrix. Even the truncation of the smallest matrix elements to zero can introduce large errors in the results. This is because every non-zero value represents a coupling action of one DOF with the load on another DOF. A break in the continuity of such actions within a model often produces disasterous results. To represent a dynamic range of 10^5, at least 17 bits are required; $2^{17}=1.31 \times 10^5$. Our processor described in Chapter 4 uses 32 bits, which allows a dynamic range of $2^{32}=4.29 \times 10^9$ to be represented.

3.6 Summary and Conclusion

A standard plate bending finite element has been used to discretize and model an aluminum plate. The structure is simple but very appropriate for an initial case study in the application area of finite elements. The problem formulation resulted in a structure stiffness matrix that demonstrates all the expected properties of a finite element problem. Specifically, the dynamic range of the stiffness matrix is five orders of magnitude, requiring a processor with at least 50 dB of dynamic range. Thus, the need for a processor with many bits of accuracy was demonsrated by our simple case study.

4. Optical Linear Algebra Finite Element Processor

4.1 Introduction

This chapter will describe an optical processing system suitable for the solution of finite element and banded matrix systems of equations. High-speed optical processing is combined with digital data encoding to yield accurate and fast solutions.

First, the limitations of proposed analog processors will be discussed (section 4.2). Next, the operation of our processor is described in general terms, and its performance is evaluated (section 4.3). Fabrication of a specific system is discussed in terms of presently available components (section 4.4), and the algorithm for the solution of finite element problems is then presented (section 4.5). It will be shown how a direct solution can be implemented with only one channel of the processor, and other performance characteristics will be mentioned (section 4.6). Finally, the problem of very large systems of equations is addressed (section 4.7).

4.2 Analog Optical Processors

Many optical processors have been proposed [15-18] to compute matrix-vector or matrix-matrix products. Most of these represent each number as one analog signal, and perform multiplications accordingly. One such architecture is the basic frequency-multiplexed optical matrix-matrix systolic array processor [18]. A schematic diagram of the processor is shown in Figure 4-1.

Many types of matrix-matrix and matrix-vector manipulations may be implemented on this processor. The most basic application, a matrix-matrix product, can be described by considering the following equation

$$AB = C \tag{4.1}$$

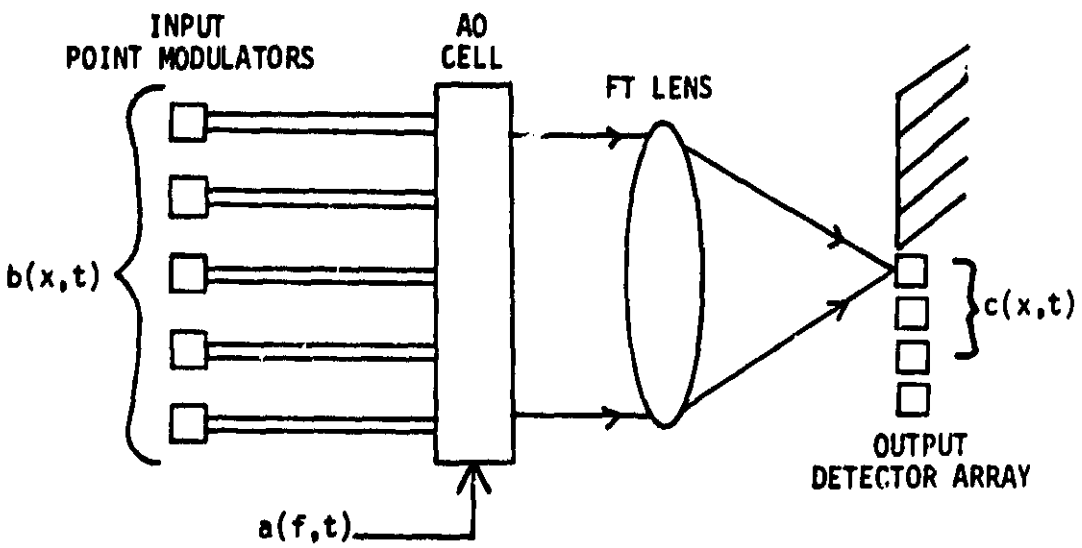


Figure 4-1: Frequency Multiplexed Optical M-M Processor

As indicated in Figure 4-1, the M point modulators (laser diodes or a multichannel AO cell) are imaged through M regions of the acousto-optic (AO) cell (each region separated in time by T_B). After leaving the cell, the entire light distribution is collected (summed) by the Fourier Transform lens and focused onto an output linear detector array. Light corresponding to each information frequency in the AO cell focuses onto a different detector in the output array.

For N by N matrices, (2N-1) point modulators are required. They are fed with the elements of B, encoded in space x and time t as b(x,t). The point modulators are fed with one row or column of B every bit time T_B. The N point modulators used for the b(x,t)'s are shifted by one each T_B, beginning with the lower N point modulators at the first T_B.

The corresponding columns or rows of A are fed into the AO cell encoded in frequency f and time t as a(f,t). There are N frequencies used so that the entire A matrix can be present in the cell. As the a(f,t) data propagates through the cell, the b(x,t)'s are applied to N different point modulator locations, thus tracking the propagating a(f,t) data in the AO cell. Thus, a b(x,t) vector at the input is multiplied by the a(f,t) matrix in the cell, and the matrix-vector product,

Ab=c, vector appears on the detector array every T_B. With a new column of B entered each T_B, after a time NT_B, the matrix-matrix product in (4.1) is produced as c(x,t). By recycling the matrix data in the cell, only N point modulators are required. By pulsing the point modulators faster than once each T_B, the matrix product can be produced in less time.

The system just described, and similar analog optical systems are very attractive for obvious reasons. Unfortunately, the accuracy of such analog systems is limited by the linear dynamic range of the components used, optical and electrical noise in the systems, and by practical alignment capabilities. The dynamic range of the components is the major limitation, and their performance will probably not improve significantly in the forseeable future. AO cells have a linear dynamic range of 30-40 dB, and multi-channel AO cells are also limited by optical, acoustic, and electrical crosstalk. Detector arrays typically have linear dynamic ranges (at useable speeds) of a few thousand to one (although individual detectors can achieve 50 dB dynamic ranges). Point modulators (especially multi-channel AO cells) also typically have a 30-40 dB linear dynamic range, although some new laser diodes can achieve a 50 dB dynamic range.

In the most optimistic scenario, an analog optical processor could be fabricated with a linear dynamic range of 30-40 dB. This translates into about 9-12 bits of accuracy. Twelve bit accuracy is not useful for most significant scientific calculations. If an optical system could be built with 50 dB dynamic range, it would only provide about 16 bits of accuracy. This might be sufficient for some calculations, but it does not compare with the 32-bit accuracy easily achievable on most digital systems. Even more important, 50 dB dynamic range is still insufficient for most significant scientific calculations (including large finite element problems). As demonstrated by the case study results in section 3.5, an optical processor is needed with much more accuracy than analog optical processors for the solution of finite element problems.

One obvious solution to this problem is to represent data digitally in an optical system. The digital encoding may take place in one of three available dimensions: space, time, or frequency. In each case, the linear dynamic range requirements are greatly reduced. For binary encoding, only two analog levels need to be represented in the processor (0 and 1), and thus a dynamic range of only 3 dB is required. Such encoding is obtained at the expense of increased size and decreased speed of the processor. However, such trade-offs are reasonable considering available components. The use of multiple levels (not just two) for encoding is another preferable alternative to speed and complexity trade-offs, and it will be discussed when appropriate. In the optical processor we will consider, fixed point representation is used. Approaches to implementing floating point operations optically is a subject for future research. Our proposed optical processor architecture is described in the next section.

4.3 Digitally-Encoded Optical Processor Architecture

Two binary-encoded numbers are easily represented in an optical processor, as mentioned in the previous section. Multiplication of these binary encoded signals to produce a meaningful product can be achieved by forming the convolution of each number's bits as we now discuss.

Consider the multiplication of two 3-bit binary encoded numbers, $b_2b_1b_0$ and $a_2a_1a_0$, where each bit b_n and a_n is a one or a zero. The most familiar method of multiplying these two numbers is illustrated in Figure 4-2. This is the popular shift-and-add method. The product is obtained by multiplying each c_n value by its appropriate power of two, and adding the results together. Each c_n value is the sum of the corresponding shifted partial products in the corresponding column in Figure 4-2. The c_n's are mixed binary numbers (i.e. their value may be any decimal number). In the example shown, each c_n can have a value from 0 to 3.

Determining the c_n's in the standard shift-and-add multiplication method is the primary operation involved in computing the product. From Figure 4-2 and 4-3 we

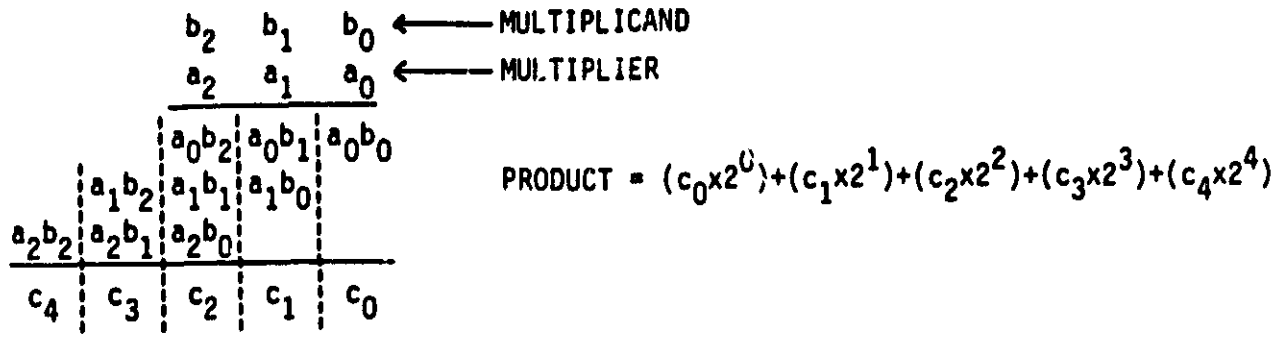


Figure 4-2: Multiplication by Shift-and-Add Method

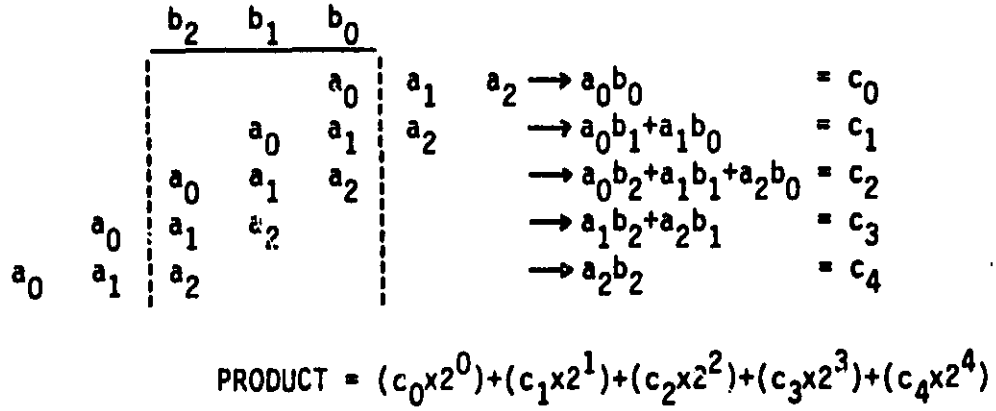


Figure 4-3: Multiplication by Bit Stream (Digital) Convolution

see that the c_n's are simply the convolution of the bits of the encoded numbers. As shown in Figure 4-3, as the two bit streams are slid across each other, multiplied and the partial products summed, the outputs at the five intervals shown are the c_n's. Since convolution is formed by reversing the order of one of the waveforms, the least significant bits (LSB's) form the first partial product. The c_n's of Figure 4-3 are identical to those of Figure 4-2.

This example can be extended to any number of bits, and any radix. Thus, multiplication of two encoded numbers can always be performed by convolving the two number's bit streams, and weighting the results by the appropriate powers of

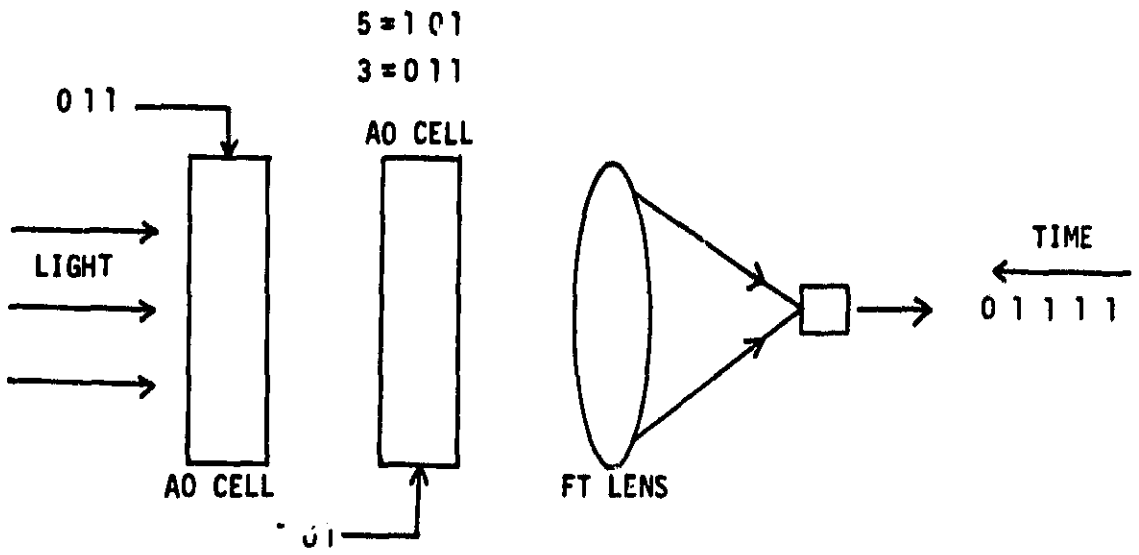
the radix used. This concept was proposed by Whitehouse and Speiser [19] for implementing fast multipliers using CCD convolvers. It was introduced to optics by Psaltis and Casasent [20].

Fortunately, convolution is an operation that is easily implemented on optical processors. One approach is to multiply the Fourier Transforms of the encoded data, and then form an inverse Fourier Transform of the product. This yields the convolution of the encoded data. Since lenses produce Fourier Transforms and Inverse Fourier Transforms, this method is very appropriate. It is detailed in [20]. However, this convolution implementation at high speed requires high frame rate spatial light modulators on which to record the Fourier Transform of one of the numbers. Such devices do not yet exist and thus other optical convolution approaches are preferable.

The preferable approach for implementing convolution on an optical processor is to use an AO cell to represent the bit stream of the multiplier and the multiplicand. Multi-channel AO cells can be used to convolve many bit streams in parallel. This method is described in [21], [22]. Reference [21] details a systolic multi-channel processor. These systems perform the convolution in Figure 4-3 in space, and the AO cells are used to provide time sequential shifting of the bit streams. An example is shown schematically in Figure 4-4 for the multiplication of the numbers 5 and 3 encoded in three binary digits.

The two AO cells are imaged onto each other. The product of different shifted bit streams is thus produced sequentially in time. The output lens forms the sum of the partial products on the detector. The data is fed, as shown, with the LSB entered first in each cell. The five convolution values obtained on the detector at successive time intervals are shown. When these are multiplied by the appropriate powers of two and added, the desired product of 15 is obtained.

A method for optically multiplying digitally encoded data using vector outer products is presented in [23]. Implementation requires the use of spatial light



$$\text{PRODUCT} = (1 \times 2^0) + (1 \times 2^1) + (1 \times 2^2) + (1 \times 2^3) + (0 \times 2^4) = 15$$

Figure 4-4: Digital Convolution With AO Cells

modulators, and thus is less attractive because of the slow rate at which such devices can be updated. Reference [24] presents an algorithm using 2's complement encoding to multiply bipolar numbers by convolution. This method requires the size (number of bits) of the product to be known a priori, and the architecture requires a 2-dimensional spatial light modulator and a 2-dimensional detector array. Thus it has practical speed and data readout problems.

The approach that we will use to implement the convolution on our processor is the shift-add method of Figure 4-2 using the architecture of Figure 4-5. This method will be shown to be very useful for banded matrix calculations.

In this example, the 5-bit binary representations of 27 and 13 are multiplied. The bits of 27, the multiplicand, are present simultaneously in the five adjacent point modulators at P_2. The bits of 13, the multiplier, are fed serially to a single input point modulator at P_1, which is imaged onto all five channels of the P_2 point modulator. P_2 is imaged onto a detector array at P_3. The contents of P_2 are thus incident on the P_3 detectors if the corresponding input P_1 bit is a 1. The contents of the P_3 detectors are shifted by one at the same rate as data is fed to P_1. Thus, the product of the data in P_2 and the current input bit to P_1 is

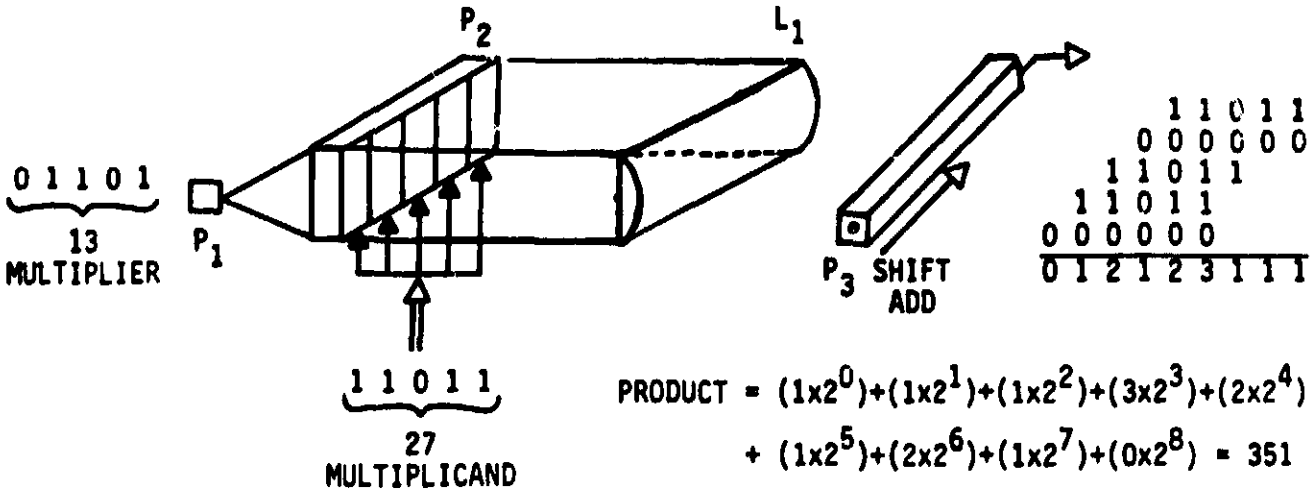


Figure 4-5: Optical Convolution of Digital Data via the Shift-Add Method

added to a shifted version of the previous input and P_2 product, and to subsequent products, by detector integration.

The detector array has five elements (as does the P_2 point modulator). The process of multiplication, shifting, and summing is repeated for all input bits fed to the P_1 point modulator. This produces a mixed binary output from the detectors that is the product as in Figure 4-2. The newest least significant bit is valid and is shifted out on each shift cycle. The mixed binary values can then be multiplied by the appropriate powers of two and added to produce the product 13x27=351, as shown in Figure 4-5.

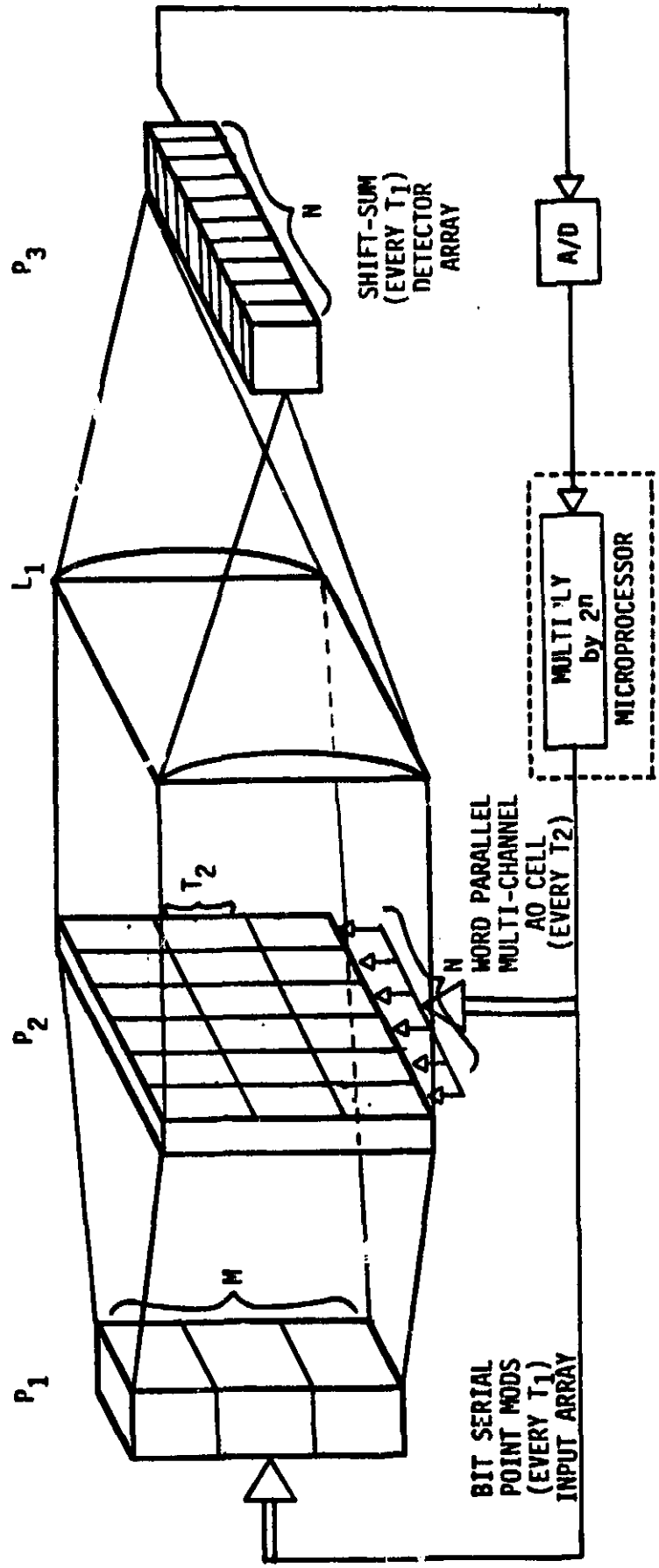
It is easy to extend this architecture to form a processor that can multiply several such numbers in parallel. To achieve this we simply use several input P_1 point modulators and divide each channel of the P_2 AO cell into several regions. Figure 4-6 shows this architecture for the parallel multiplication of three pairs of encoded numbers. With 3 separate 1-dimensional shift-and-add output detector arrays, the 3 products are produced in parallel. With one output detector array (as shown in Figure 4-6) the sum of these 3 products is produced. This latter operation is preferable as it is the one required in matrix-vector multiplication (specifically in producing a vector inner product).

A multi-channel AO cell is used at P_2 in Figure 4-6 in place of the point modulators in Figure 4-5, and a 1-dimensional array of point modulators is used for the bit serial inputs at P_1. If N bits are used in the number representations, an N-channel AO cell is needed at P_2 (five channels are shown in Figure 4-6), each channel carries one of the N bits. An array of M input point modulators is used in P_1 (three are shown in Figure 4-6). A bit is presented to each P_1 point modulator every T_1 seconds. Each channel of the multi-channel AO cell is divided into M regions. The light from each P_1 point modulator is imaged across a different region of each channel of the cell in P_2. Data is input to the N-channel cell in word parallel form every T_2 seconds, where $T_2=NT_1$. Thus, a number's bit stream, or binary word, is present in a region of the cell for T_2 seconds, then it moves into the next region of the cell. During each T_2, N bits (one every T_1) are presented to the P_1 point modulators. Thus, the light leaving P_2 in Figure 4-6 is M of the operations done in Figure 4-5.

The light leaving P_2 is collected by cylindrical lens L_1, which sums the output from each of the N AO cell channels in P_2, and focuses each onto a separate one of N detector elements in the shift-add output detector array in P_3. The output array shifts and adds the incident light every T_1. The values in the detector array thus represent the sum of M multiplications of the form in Figure 4-5. Since the data incident on the detectors is mixed-binary, any number of multiplications M may be summed. Each mixed-binary value shifted out of the detector every T_1 (once a bit is shifted out of the N-element detector array, it is valid) is analog-to-digital converted, multiplied by the appropriate power of two, and then added to the output at the next T_1.

The system thus produces the addition of M multiplications with N-bit accuracy every T_2. In other words, the processor of Figure 4-6 performs an M-element vector inner product (VIP) every T_2 (M multiplications and M-1 additions). We define the processor as having M processing, or processor, channels (as opposed to the number N of AO cell channels). A single processor channel is shown in Figure 4-5.

Figure 4-6: Binary Encoded Optical Linear Algebra Processor



We have thus shown how the processor of Figure 4-6 can perform vector inner products with N-bit accuracy. An algorithm will be presented in section 4.5 which details how this processor is ideal for solving banded matrix problems (specifically a finite element system of equations). Now we will address the issue of handling bipolar data on this processor.

The cylindrical lens L_1 of Figure 4-6 sums the outputs from all M regions of each channel of the AO cell in P_2. This presents a problem when the products leaving P_2 are bipolar. Since we have only used the magnitude of the numbers for binary encoding, the processor and thus lens L_1 can only treat signals as positive numbers (unipolar). Processing bipolar data requires specialized treatment. Some methods to achieve this are now discussed.

If it could be guaranteed that all the products leaving P_2 are of the same sign for a given T_2 (each component of the VIP is either positive or negative), a simple digital logic test can be used to determine the sign for the vector inner product. In this case, the numbers would be represented digitally with a sign bit (i.e. in sign-magnitude form), but only the magnitude bits would be entered into the processor as before. The processing simply requires calculation of the exclusive-or of the sign bits of the two numbers being multiplied. This can be performed within the microprocessor. The result must be available before the current T_2 period is over (this requirement is trivial). If the resultant exclusive-or is a 1, the sign bits of the multiplier and multiplicand differ, and the product is negative; if the result is a 0, the product is positive. If not all the products leaving P_2 have the same sign, then this method cannot be used for an M-element VIP. However, it can always be used for multiplying two numbers

If the integrating cylindrical lens L_1 is replaced by an imaging lens, and M detector arrays are used (one for each region of the cell), the above exclusive-or method may be used to determine the sign of the product produced on each detector array. The VIP would then be assembled within the microprocessor by

simply adding or subtracting the products accordingly. The drawback of this method is that M detector arrays are required in P_3, and that the additions or subtraction to form the VIP must be done digitally, rather than with a lens.

Another approach to using bipolar data on our processor is to double the size of the problem by expanding the matrix equations into a positive and a negative part, as detailed in [18]. This method requires no modification to the processor, however the processing time is doubled.

Reference [24] suggest handling bipolar data with twos complement data encoding. A similar twos complement representation can be implemented on our processor to handle bipolar data, with some special care. This process is not presented in this report due to the timeliness of its formulation. It is a subject for future work and will be presented in a forthcoming document. However, we will show how the exclusive-or test for one multiplicaltion is adequate to handle bipolar data for the direct algorithm we will implement on the processor.

The approach to bipolar handling that we will use for finite element problem processing is the exclusive-or test method. The algorithm presented in section 4.5 shows that only one processor channel is needed for finite element problems. Thus bipolar data is easily handled by comparing the sign bits of the data in P_1 and P_2 of the single processor channel that is used.

4.4 Processor Performance and Fabrication

This section will discuss performance and fabrication issues for the processor of Figure 4-6. Many high-speed optical architectures have been proposed, however very little is ever documented about how data can be realistically input to and output from these systems, since those operations must be controlled digitally. We will discuss specific analog and digital hardware systems that are capable of supporting the data throughput of our processor at good performance speeds. This hardware is being fabricated at Carnegie-Mellon University (CMU) for the

purpose of testing the laboratory performance of our proposed processor. Some general optical processor fabrication issues are discussed in [25].

To compete with the accuracy of digital computing systems, the processor should be capable of 32-bit encoding, or N=32. This requires a 32-channel AO modulator for P_2. Such a device, built by Crystal Technology, Inc., was recently purchased by CMU. No performance tests have yet been performed on the cell, but some design specifications are given below:

- 32 channels, TeO_2 longitudinal mode.
- 5 microsecond aperture time
- Channel-to-channel crosstalk better than 30 dB over 3 microseconds.
- 200 Mhz bandwidth centered at 400 Mhz or less
- Operation at λ=633 nm or λ=820 nm.

The 32-channel cell specifications are used to set some performance measures for the processor. We will consider a processor with 10 input point modulators in P_1, or M=10. This is a realistic fabrication level. A larger P_1 input array would create a large system anamorphism, since the 32-channel cell aperture is only a few centimeters long. Specific input array devices are discussed later.

We will assume that all of the 5μs aperture of the 32-channel cell is useable. Since there are M=10 point modulators, each channel of the cell is divided into 10 regions, each region with an aperture of 0.5 μs. The period T_2=0.5 μs is the time it takes the word parallel data to propagate through a region of the cell, i.e.

$$10 \times T_2 = 5\mu s = T_{aperture} \tag{4.2}$$

During each T_2, each input point modulator is pulsed on with the N bits of an input word sequentially every T_1. Since N=32,

$$32T_1 = T_2 \tag{4.3}$$

and from (4.2) and (4.3),

$$10 \times 32 \times T_1 = 5\mu s \qquad (4.4)$$

Solving for T_1 yields T_1=15.625 nsecs. Thus, to run the processor with N=32 bit accuracy, binary encoding, and M=10 input point modulators, data must be fed to the point modulators, and shifted and added in the detector array, at $1/T_1$ or 64 Mhz. We now discuss processor operation and input and output circuitry at this speed.

The processor performs a 10-element VIP every T_2, or 0.5 μs. A 10-element VIP consists of 10 multiplications and 9 additions. Thus, the processor computes 2×10^7 multiplications and 1.8×10^7 additions per second. This optical computation rate is not as fast as many proposed optical processors, but two things must be considered: 1) this processor computes with 32-bit accuracy; 2) we will document realizable circuitry to input and output processor data, unlike many high speed optical processor proposals.

The P_1 input point modulator array can be fabricated with three types of devices. Each will be mentioned here, a more detailed discussion about fabrication is in [25]. An array of point modulators could be built using separate LED's or laser diodes (LD's). Such an array could use graded index (GRIN) optical elements to couple the light from each source to a point, for input to fiber optic interconnections. The fiber optics terminate in another set of GRIN elements. The second set provides tightly packed separate collimated sources. These arrays provide dense, low divergence sources, however, they are difficult to fabricate, and single-mode fibers are difficult to align.

Another point modulator array alternative is an array of collimating pens. A collimating pen is a laser diode with individual collimating optics. Several collimating pens are commercially available, making their use attractive. Reducing optics is required to produce densely packed beams covering the 2.1 cm, 5 μs aperture of the 32-channel AO cell at P_2.

The best point modulator array choice for our processor is a multi-channel acousto-optic point modulator cell. These devices have a very low beam divergence angle. They require magnification or demagnification optics, but only one device needs to be aligned. Multi-channel AO cells, as discussed earlier, pose some limitations for analog systems. Specifically, electrical, acoustic, and optical isolation between the channels. However, for digitally encoded systems such as ours, where dynamic range requirements are greatly reduced, these devices are very appropriate.

Since M=10 in our processor, a 10-channel AO cell is needed for an input array. A 10-channel cell appropriate for point modulator use (small aperture time), was recently purchased by CMU from Crystal Technology, Inc. Some specifications are given below:

- 10 channels, TeO_2 longitudinal mode.
- Point modulator operation (less than 2 microsecond aperture time).
- Channel-to-channel crosstalk better than 30 dB.
- 80 Mhz center frequency
- Operation at λ=633 nm.

We now describe the input circuitry that will be used to feed data to the point modulators at 64 Mhz. At CMU, we are currently assembling a system that employs parallel high-speed buffers, with access times of 100 nsecs. The system has 6 boards, with 8 memory channels per board, each 12 bits wide and 4K (words) long. With a 100 nsec access time, each 12-bit word in a channel can be updated at a rate of 10 Mhz. An ECL multiplexer can be used to scan 10 of the bits of a channel at 100 Mhz, providing a 100 Mbit output. Since 100 nsecs are required to scan 10 bits at that rate, the channel data can be updated when the next 10-bit multiplexer scan is ready. The scanning rate can be reduced to achieve 64 Mhz or any other desired speed below 100 Mhz. Thus, 10 memory channels would be needed to drive the M=10 point modulator array, and there are 48 available.

The digital system is completely programmable, and in operation, the data will be downloaded from main memory into the parallel high-speed buffers. The buffers, 4K words long, will be reloaded as the current data is being fed to the processor. Thus, we will operate the system in a burst processing mode, where the processor runs at full speed, but data flow can be stopped to unload and reload the buffers. This provides an efficient method of testing the processor at the full data flow rate.

The dynamic range requirements of the processor, through P_1 and P_2 (see Figure 4-6), are quite low. If binary encoding is used, only two levels need to be represented in the point modulators at P_1, and the multi-channel AO cell at P_2. Binary encoding translates into a dynamic range requirement of 3 dB. This isolation level is easily achievable in the multi-channel AO cells of P_1 and P_2 (see the 32-channel and 10-channel AO cell specifications earlier in this section).

Fabrication of the shift-add detector array of P_3 is not trivial for a 64 Mhz shift rate. A linear CCD detector array is an obvious first consideration. Each detector element in the array produces an amount of charge proportional to the intensity of the light that is incident on it. This charge is transferred and added to a CCD analog shift register, which shifts the packets of charge from one element to the next. The charge is usually serially output at one end of the shift register. This output could be fed into a single analog-to-digital converter, as shown schematically in Figure 4-7.

Since N=32 in our processor, a 32 element detector array is needed. These detector arrays are commercially available with 64, 128, 256, 512, etc., elements. However, the maximum sample rate for these devices is 10 Mhz, with 20 Mhz promised in the near future. At these rates, the dynamic range of the detectors is severely limited. At 10 Mhz, dynamic ranges of only 100:1 are possible, with some improvement if the detector array is cooled and if a lot of light is used. Useful devices shifting at 64 Mhz do not seem feasible in the forseeable future, with the possible exception of some GaAs devices.

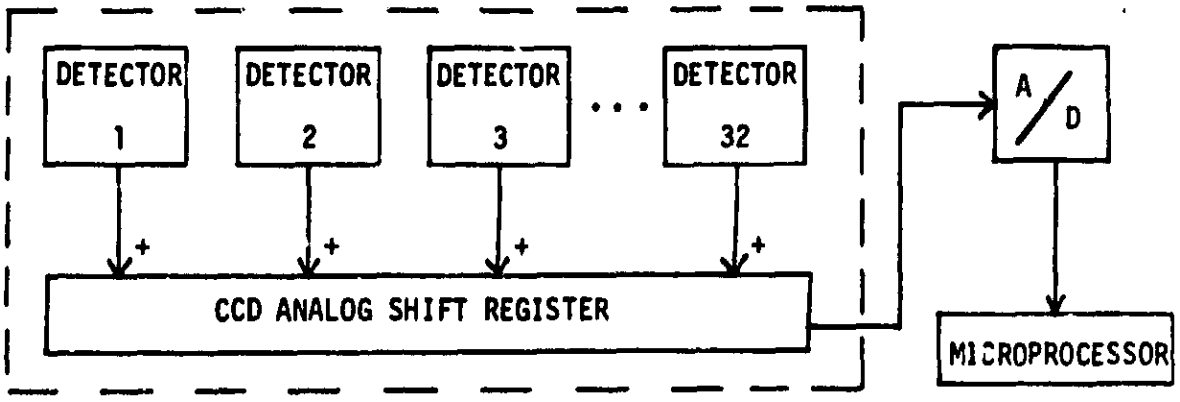


Figure 4-7: Linear CCD Detector Array Configuration

For binary data encoding, general purpose dynamic range requirements can be specified for the detector system. If a linear CCD detector array is used, each CCD analog shift register element needs a dynamic range of 320, since a maximum value of M=10 can occur on a detector, and there are 32 detector elements. Actually, the shift element register for detector 1 needs a dynamic range of 10, the register for detector 2 requires 20, etc., and the register for detector 32 needs a dynamic range of 320. Since the linear CCD detector array is fabricated uniformly, a dynamic range of 320 (or 25 dB) is required. This is difficult at 64 Mhz. It may be possible with GaAs technology, but such devices are not yet available.

The single A/D converter in Figure 4-7 must convert 320 levels. Thus, a 9-bit analog-to-digital converter is required. However, the fastest 9-bit A/D converter commercially available runs at only 20 Mhz. Thus, the detector array configuration in Figure 4-7 is not very useful for our purposes.

In order to operate at 64 Mhz, the detector array configuration we will use consists of N=32 individual detectors, 32 A/D converters, and 32 high-speed ECL registers. The arrangement is shown schematically in Figure 4-8.

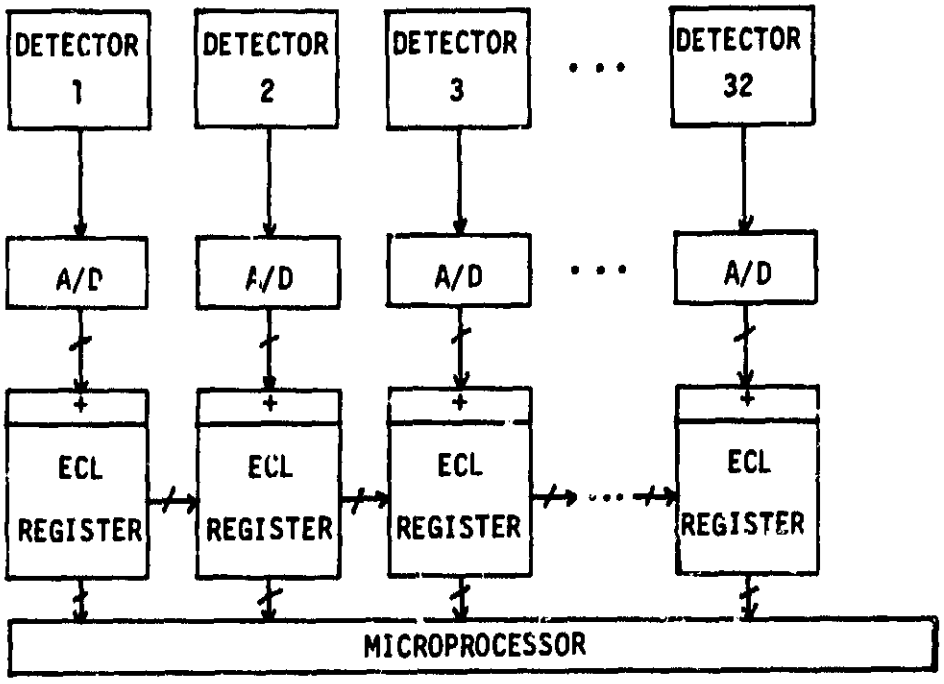


Figure 4-8: Individual Detector/ECL Detector Array Configuration

This configuration uses parallel rather than serial output from the 32 detectors. The output from each individual detector is A/D'ed every T_1, at 64 Mhz. The output from each A/D is added to the contents of a digital ECL register, which includes addition logic. The ECL circuitry shifts the contents of each register to the next register every T_1. With ECL circuitry, 64 Mhz shift rates are easily achievable. This configuration requires N=32 A/D's and N=32 registers, but it avoids the dynamic range and speed problems of using a linear CCD detector array.

The detector arrangement of Figure 4-8 has realistic requirements, and it was chosen for our proposed processor. Each detector requires a dynamic range of [M=10]+1 (1 for the zero level). Thus, each A/D must convert 11 levels, which requires four bits. Individual detectors can operate at 64 Mhz with 11 levels, or about 10 dB of dynamic range, quite easily. Four bit A/D converters are commercially available at speeds of 100 Mhz. Thus, this detector arrangement can be built for 64 Mhz operation with existing electronics.

We now consider the case of encoding in a radix other than two. If radix R is chosen, and the detector array of Figure 4-8 is used, each detector would require a dynamic range of $10x(R-1)^2+1$, since (R-1) is the maximum level possible in P_1 and P_2 and they could multiply each other, plus one for the zero level. Each A/D converter would require $LOG_2[10x(R-1)^2+1]$ bits, rounded to the next largest integer. The larger the radix R, the larger the dynamic range requirements become. However; with a larger R, fewer than 32 channels and encoded bits are required to obtain 32-bit (radix two implied) digital accuracy. Also, processing time is faster because less than 32 T_1's are required for each T_2. Likewise, use of 32 channels and encoded bits with an R greater than two would result in a computational accuracy greater than 32-bits, with the same processing speed. These issues will be considered in more detail in future work.

There is another requirement of our processor that makes the individual detector/ECL detector array configuration more attractive than a linear CCD detector array. At the end of every T_2, there are still N mixed binary values present in the detector array. If a standard CCD array was used, these N values would have to be serially shifted out and A/D'ed, causing a delay in the processing because T_2 would need to be longer than NT_1. If the CCD analog shift register could be emptied in parallel to another CCD register, processing could proceed without delay. However, such parallel output devices do not exist. When the detector configuration of Figure 4-8 is used, the N values are A/D'ed in parallel, and the contents of the ECL registers can be transferred to the microprocessor in parallel via the digital hardware. Thus, no delay exists when the arrangement of Figure 4-8 is used.

A final comment should be made about the operation of our proposed processor in Figure 4-6. As explained earlier, the word parallel data fed into the N=32 channels of the AO cell in P_2 moves through one of the M=10 regions of the cell every T_2. Within each T_2, the word parallel data in each region is multiplied by N=32 bit serial values from the corresponding input point modulator.

If the word parallel data was input to the multi-channel cell as continuous data, there would be an overlap of data in each of the M=10 regions, within the T_2 periods. This, of course, is unacceptable. Thus, the data cannot be input continuously, but must be fed to the cell in pulses. Specifically, each pulse must be T_1 long or less to avoid any overlap into the next region. If each pulse is exactly T_1 long, it will enter a region at the start of a T_2 period, and leave and move into the next region exactly at the start of the next T_2 period. Thus, light will not be modulated by a signal in the full 21/10=2.1 mm of a cell region, but rather by a signal 2.1/N=32 = 0.066mm long. Thus, this represents a reduction of light intensity leaving P_2 by a factor of 1/(N=32).

4.5 Finite Element Processing Algorithm

It has been shown in section 4.4 that the optical processor of Figure 4-6 will compute an M-element VIP every T_2 with digital encoding and digital accuracy. The architecture is very general purpose, and can be used to implement many matrix-matrix and matrix-vector manipulation algorithms.

The focus of this research is to solve systems of linear algebraic equations (that arise from finite element analysis) on an optical processor. As we have seen in Chapter 2, the finite element equations in matrix form yield a well-banded matrix when formulated properly. It is appropriate to exploit this bandedness when implementing a solution algorithm on the processor, and we will show how the optical processor of Figure 4-6 is well suited for a band oriented algorithm. First we will show, in general, how the computation of a banded matrix-vector product may be implemented on the processor. Then we will detail an algorithm for solving finite element equations which involves many banded matrix-vector products.

4.5.1 Banded Matrix-Vector Multiplier

To illustrate how a banded matrix-vector product can be implemented on our optical processor, we will schematically represent the processor without the digital encoding. Thus, for clarity, the data flow will be explained as if an analog processor was used. This in no way limits the performance of the digitally encoded processor, since the same architecture is used in terms of the multiplications. The time between VIP's, T_2, is often called the bit time, or T_B, in analog processor descriptions. Thus, in the data flow explanations below, the digital encoding is supressed, and T_B is effectively the same as T_2.

Consider the banded matrix-vector equation

$$\mathbf{Ab} = \mathbf{d} \tag{4.5}$$

where **A** is a banded matrix. This equation is expanded in the top part of Figure 4-9 for the case of a 7 by 7 matrix **A** with a bandwidth of 5. The values of **A** and **b** are known, and the desired matrix-vector product is the vector **d**. The components of **d** can be obtained by the algorithm illustrated in the lower part of Figure 4-9. This algorithm is similar to one described in [17]. The processor requires a number of point modulators equal to the bandwidth of the matrix. As shown in Figure 4-9, each diagonal of the matrix is fed sequentially into one of the point modulators (i.e., one row of the matrix is fed in parallel to the point modulators).

The squares in Figure 4-9 represent the 5 required point modulators, P_1 of Figure 4-6, and the long rectangle represents the acousto-optic cell, P_2 of Figure 4-6. Note that only the inputs and outputs for the first 6 bit times, T_B, of the algorithm are illustrated in the figure. The components of the vector **b** are fed to the AO cell, and the band elements of **A** are fed to the point modulators with proper timing. Note that the upper diagonal is input to the lower (first) point modulator, the main diagonal to the middle point modulator, and the lower diagonal to the upper (fifth) point modulator. A new row of **A** elements is input every T_B,

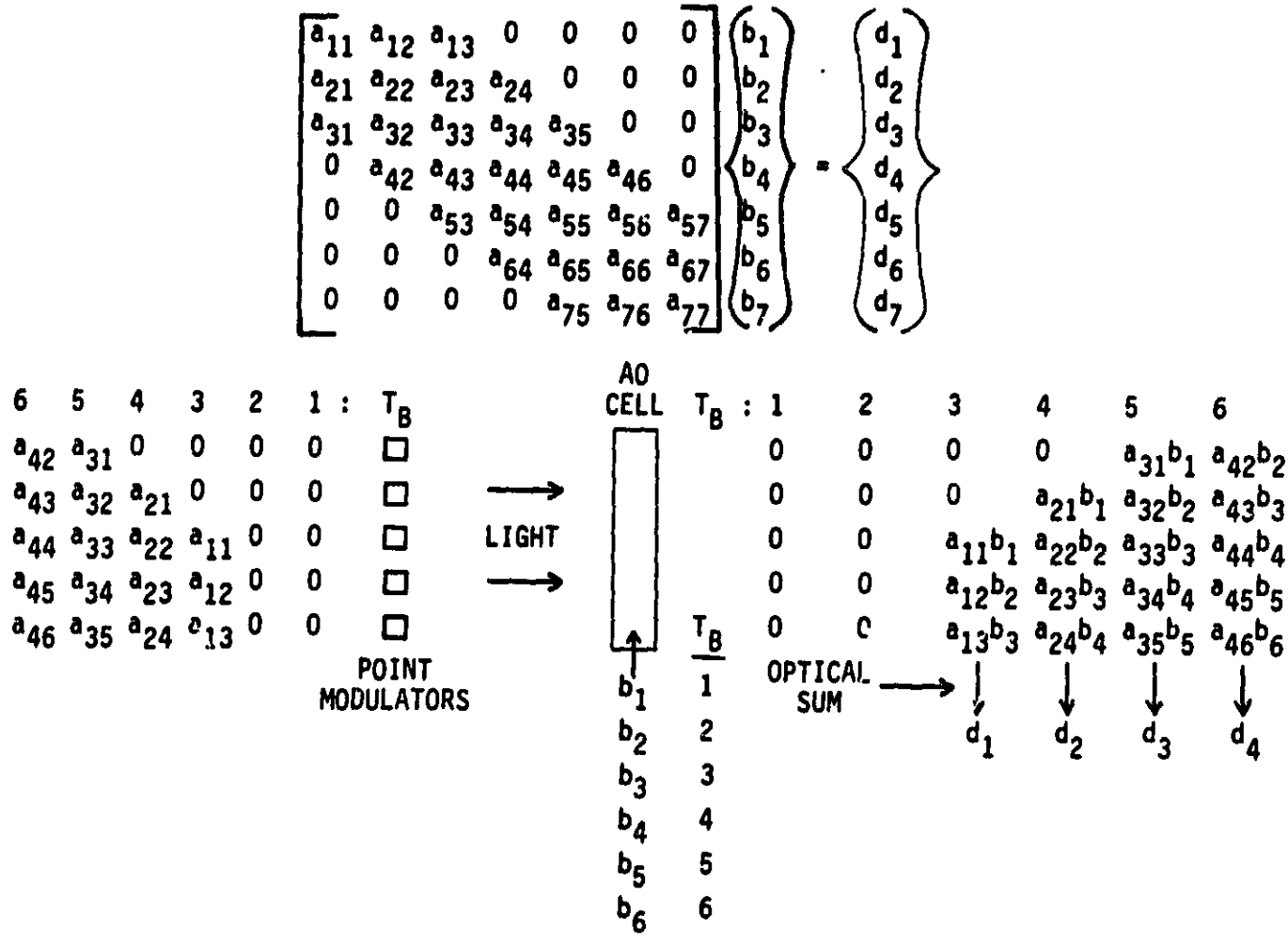


Figure 4-9: Banded Matrix-Vector Product Algorithm

and the b elements move to the next point modulator's region of the cell every T_B. As the b elements propagate through the AO cell, the proper A elements are fed to the point modulators to form the required partial products at the output. The outputs leaving the 5 regions of the AO cell at P_2 at each T_B are shown at the right in Figure 4-9.

In Figure 4-9, the bit time indices above the columns represent the bit time when those inputs are applied to the point modulators, and when those outputs are present as light leaving the cell. The indices next to the b conponents represent the bit time when that component enters the cell (i.e. is in the region

of the lower (first) point modulator). It can be seen from the figure that the partial products formed by the system, when added together at each bit time, sum to one of the **d** components. Specifically, d_1 is formed first, and d_7 is formed at the last bit time.

Thus, the algorithm illustrated in Figure 4-9 may be used to compute any banded matrix-vector product. The algorithm is very attractive since only the elements of **A** within the band are processed, eliminating time consuming processing of the non-band zero elements. The illustration in Figure 4-9 is simply an analog schematic of the processor of Figure 4-6. Physically, the summation of the partial products, output from the system of Figure 4-9 every T_B, is performed by the cylindrical lens L_1 in Figure 4-6, to produce the components of **d**. Again, the bit time T_B in this explanation corresponds to time T_2 in our digitally encoded processor of Figure 4-6. The digital encoding is simply manifested in the N T_1's which equal T_2, and the extra dimension of the N-channel AO cell.

4.5.2 Direct LU Decomposition

Finite element equations may be solved by direct or indirect (iterative) solution methods, as mentioned in Chapter 2. Since direct solution methods are more popular, implementation of a direct solution method on our optical processor will be examined. However, indirect solution methods have some advantages and are equally applicable for this optical processor. This subject will be discussed further in Chapter 6.

The goal of all direct solution methods is to reduce the full system of equations to a triangular system. Consider

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.6}$$

where **x** is the unknown vector. The matrix **A** can be decomposed into the product of a lower **L** and an upper **U** triangular matrix. This is the well known LU matrix decomposition. Equation 4.6 can then be expressed as

$$LUx = b \tag{4.7}$$

Equation 4.7 can easily be solved by first solving

$$Ly = b \tag{4.8}$$

for y by forward substitution, and then solving

$$Ux = y \tag{4.9}$$

for x by back substitution.

Since L and U are triangular matrices, the forward and back substitution processes are trivial, even for very large matrices. Both operations can be easily and quickly performed in digital hardware. The computational burden is performing the LU decomposition. We will use the optical processor of Figure 4-6 to perform the triangular decomposition of finite element systems of equations.

Many types of direct algorithms and decompositions exist. We will use LU decomposition since it is easily implemented on our optical processor. A symmetric Cholesky LL^T decomposition could also be implemented, but we will show that it requires extra work later in this section. The LU matrix decomposition algorithm we will use was first described for an optical processor in [27]. The algorithm may also be found in [32] and other linear algebra texts.

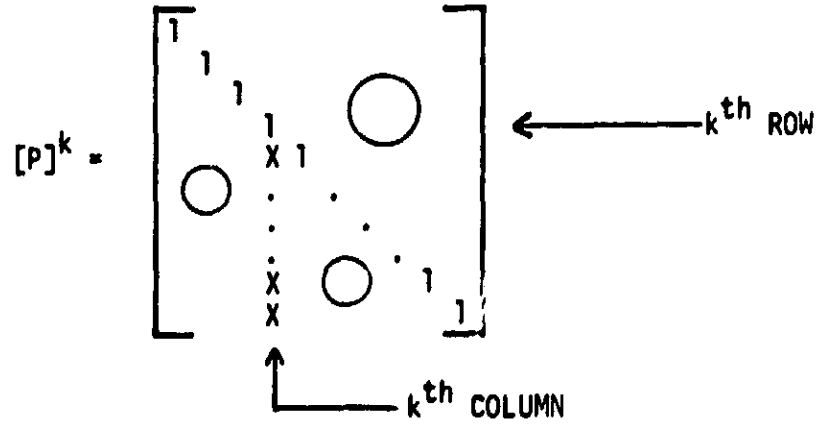
Consider equation 4.6. For an N by N matrix A, N-1 steps are required to produce a triangular matrix equation. Each step involves a matrix-matrix multiplication. The matrix A^k, where k is the step index and $A^1=A$, is premultiplied by a decomposition matrix P^k, to form A^{k+1} at each step k, i.e. the process is: form $A^{k+1} = P^kA^k$ for k=1, ..., N-1 , where P^k is a lower triangular matrix. It is a unit diagonal matrix with non-zero off-diagonal elements only in column k. Column k of P^k can be written as the column vector c^k, where

$c_i^k = 0$ for $i<k$, i.e. zero's above the diagonal

$c_i^k = 1$ for $i=k$, i.e. one on the diagonal

$c_i^k = -a_{ik}^k/a_{kk}^k$ for $i>k$, i.e. below the diagonal

Thus, the c^k values are calculated from the A^k matrix elements. It should be noted that a necessary condition for this algorithm is than no main diagonal element of A is zero. This condition is guaranteed if the matrix A is positive definite or diagonally dominant. In conventional uses of the algorithm, a zero (or small number for accuracy purposes) on the main diagonal would require pivoting, which cannot be done with our implementation of the algorithm. Thus, the matrix P^k has the form



After step 1, the first element is the only non-zero element in the first column of A^2. After step 2, the first element in the first column, and the first two elements in the second column are the only non-zero elements in the first two columns of A^3. This pattern continues, where the first k columns of A^{k+1} are upper triangular. After N-1 steps, A^N is the upper triangular matrix U. The final matrix is thus

$$A^N = A^{k+1} = P^{N-1} \ldots P^2P^1A^1$$

where

$$A^1 = A \tag{4.10}$$

Upon letting

$$P = P^{N-1} \ldots P^2P^1 \tag{4.11}$$

and noting that

$$A^N = A^{k+1} = U \tag{4.12}$$

is an upper triangular matrix, we can write

$$A^N = PA$$

or

$$U = PA \tag{4.13}$$

To show that this is LU decomposition, we can rewrite (4.13) as

$$P^{-1}U = A$$

or

$$LU = A \tag{4.14}$$

where we explicitly note that P^{-1} is a lower triangular matrix **L**.

To form the Cholesky decomposition factor L^T, where

$$A = LL^T \tag{4.15}$$

an extra step is required, as explained in [27]. From **U**, the diagonal matrix **D** is computed, where

$$d_{ii} = 1/\sqrt{u_{ii}} \quad , \; i=1, \ldots, n \tag{4.16}$$

and the matrix L^T is formed by the following matrix-matrix multiplication

$$L^T = DU \tag{4.17}$$

Equation 4.6 can now be solved from (4.14) by performing the forward and back subtitution of (4.8) and (4.9). However, it is much easier to perform the forward substitution while **A** is being decomposed. This simply involves premultiplying **b** by each P^k at each step, producing a new vector **b'** in the same N-1 steps as

$$b' = Pb \tag{4.18}$$

The same result is achieved if the augmented matrix **A|b** is multiplied by the P^k

matrices to begin with, which yields the augmented matrix U|b' after N-1 steps. This requires an extra matrix-vector product at each step.

The algorithm decomposes the original equation, **Ax=b**, into a triangular system as follows. First both sides are multiplied by **P**,

$$\mathbf{PAx} = \mathbf{Pb} \tag{4.19}$$

This requires N-1 steps (matrix-matrix products). Equation 4.19 is then

$$\mathbf{Ux} = \mathbf{b}' \tag{4.20}$$

This triangular matrix equation, (4.20), is then easily solved on a digital system. Optical solutions are possible and are detailed elsewhere [26].

The basic finite element equation, **Kd=p**, with boundary conditions imposed, is of the same form as (4.6). It meets the requirement of being positive-definite, and thus can be solved by the above algorithm.

4.5.3 Processing Time for LU Decomposition

We have seen how a banded matrix-vector product can be implemented on our optical processor. Equation 4.19 shows that the LU decomposition algorithm is composed of N-1 matrix-matrix multiplications, **PA**, and N-1 matrix-vector multiplications, **Pb**. Each matrix-matrix multiplication consists of N matrix-vector multiplications. Thus, the entire algorithm consists of N(N-1) plus (N-1), or N^2-1 matrix-vector multiplications. By examining the definition of c_i^k for i>k, it is obvious that each matrix-vector product (**P** times a vector of **A**, or **P** times **b**) involves a banded matrix with the same bandwidth as **A**. This occurs because wherever a_{ik}^k is zero, c_i^k will be zero from the definition of c^k.

In fact, $\mathbf{P}^k$ has a bandwidth equal to the semibandwidth of **A**, since P^k is strictly lower triangular. A banded **A** matrix is shown with its corresponding $\mathbf{P}^k$ matrix in Figure 4-10, where an X represents a potentially non-zero matrix element. Thus, the banded finite element matrix equation, (2.18), can be solved on the proposed optical processor utilizing the above LU decomposition algorithm, and

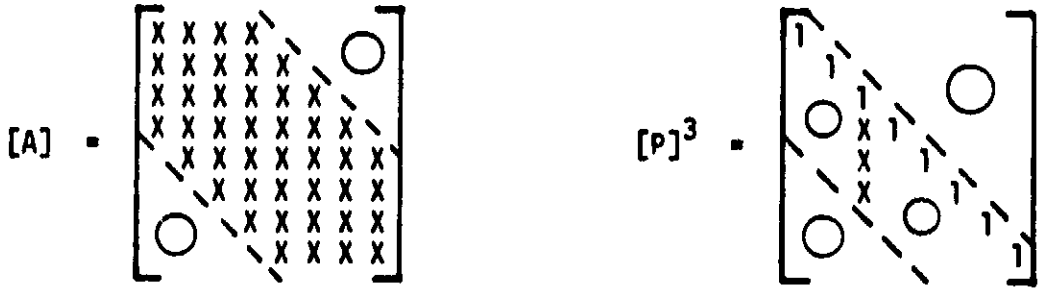


Figure 4-10: Decomposition Matrix Structure

requiring N^2-1 banded matrix-vector multiplications, which is $N(N^2-1)$ vector inner products (VIPs).

The actual number of required vector inner products is less than $N(N^2-1)$. This occurs since at each step $P^kA^k=A^{k+1}$ in calculation (4.10), the size of the matrices P^k and A^k can be reduced by one. This is obvious from the structure of P^k and realizing that the first k-1 rows and columns of A^k are not affected when A^k is premultiplied by P^k. Thus, a system of equations of size N-k+1 must be solved at each step k. Step one requires an N by N matrix-matrix (N^2 VIPs) and matrix-vector multiplication (N VIPs); step N-1 requires a 2 by 2 matrix-matrix (4 VIPs) and matrix-vector (2 VIPs) multiplication.

The number of VIPs required to perform the LU decomposition of an N by N system of equations will now be quantified. At step 1, N^2 VIPs from the matrix-matrix multiplication (4.13) are computed, and N VIPs from the matrix-vector multiplication (4.18) are computed. At step 2, the number of VIPs computed is reduced to $(N-1)^2$ and (N-1). At step k, $2^2=4$ and 2 VIPs are computed. Thus, the following formula gives the number of VIPs required for an N by N system.

$$[N^2+(N-1)^2+\ldots+2^2] + [N+(N-1)+(N-2)+\ldots+2] = \#\text{ of VIPs} \simeq N^3/2 \quad (4.21)$$

Evaluating (4.21) for N=24, 4899+299=5198 VIPs are required for the LU decomposition algorithm. The processing time required for this is $5198 \simeq N^3/2$ T_2's (the $N^3/2$ approximation is valid for large N). A problem size of N=24 represents our case study with boundary conditions imposed, where the boundary conditions specify that all the DOFs on two connected edges of the model (7 nodes, 21 DOFs) are fixed to zero. This is a common support condition.

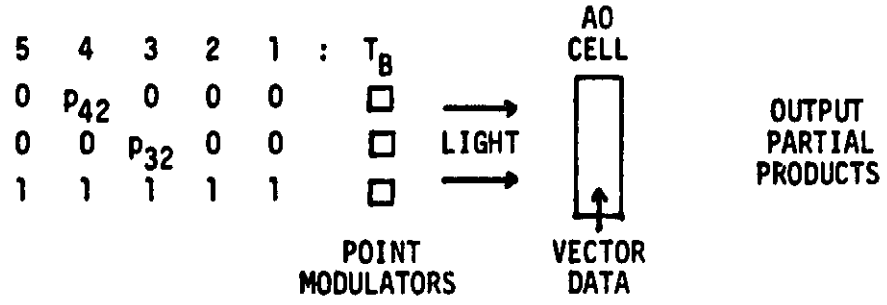
4.5.4 One Channel LU Processor

We implement the LU decomposition algorithm on the optical processor by forming $\mathbf{P}^k\mathbf{A}^k$ with N-k+1 matrix-vector multiplications, separating $\mathbf{A}^k$ into N-k+1 vectors. From Figure 4-9, the matrix diagonals are fed into each point modulator. Thus, with our implementation of the algorithm, only two point modulators are being input with non-zero data at any time. This drastically reduces the detector dynamic range requirements, which will be quantified in section 4.6.

To see this, we illustrate a specific step of the LU decomposition in Figure 4-11a. We consider the decomposition matrix $\mathbf{P}^2$ for a 5^{th} order system with a semibandwidth of 3. Since the first element of each diagonal is input to the point modulators at successive bit times, it is evident from the data flow diagram in Figure 4-11b, that at any bit time, at most only two point modulators have non-zero input data: the main diagonal input, which is always a 1, and one other input. We utilize this in section 4.6.

The formation of the elements of the $\mathbf{P}$ matrix can be easily done with dedicated hardware. Only a new $\mathbf{c}^k$ vector needs to be computed at each step. As defined above, the c_i^k elements, where i>k (the only elements that change from step to step), are formed by calculating the reciprocal of a_{kk} and subsequent simple multiplications of two $\mathbf{A}^k$ matrix elements. The analog optical processor using the LU decomposition algorithm as detailed in [27] uses analog electronics to generate the new $\mathbf{c}^k$ vector elements. Our system will use dedicated digital hardware to perform the division. Once each column of the $\mathbf{A}^{k+1}$ matrix is

$$[P]^2 = \begin{bmatrix} 1 & & & & \\ 0 & 1 & & & \\ 0 & p_{32} & 1 & & \\ 0 & p_{42} & 0 & 1 & \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (a)$$



(b)

Figure 4-11: Illustration of Only Two Non-Zero Inputs in LU Decomposition Algorithm Implementation

computed (from P^k times a column of A^k), the c^{k+1} elements can easily be computed in enough time for the next step of the algorithm.

4.6 Finite Element Processing

This section will refine the processing details specifically for solving finite element problems with the direct LU decomposition algorithm. Specifically, we will show how only one channel of the processor of Figure 4-6 is required. Comments on dynamic range requirements for the detector array are given for this implementation.

In the previous section, Figure 4-11 illustrated that only two point modulators are input with non-zero data during any T_2 (T_B, bit time) for our implementation of the LU decomposition algorithm. It was also shown that one of the point modulators is always input with a 1, and it is always the same point modulator.

The other non-zero input could be at any one of the other inputs, but if it is at point modulator m at the i^{th} bit time, it is at point modulator m+1 at the $(i+1)^{th}$ bit time.

It would not be very practical to use an M=10 input processor when solving finite element problems exclusively with our LU decomposition algorithm, because only 2 inputs are needed. However, since the input to one of the required point modulators is always 1, only M=1 point modulator is required to implement the algorithm. The one-channel processor is shown in Figure 4-12. It is one channel of the processor in Figure 4-6, with some additional electronics, and its operation is explained below.

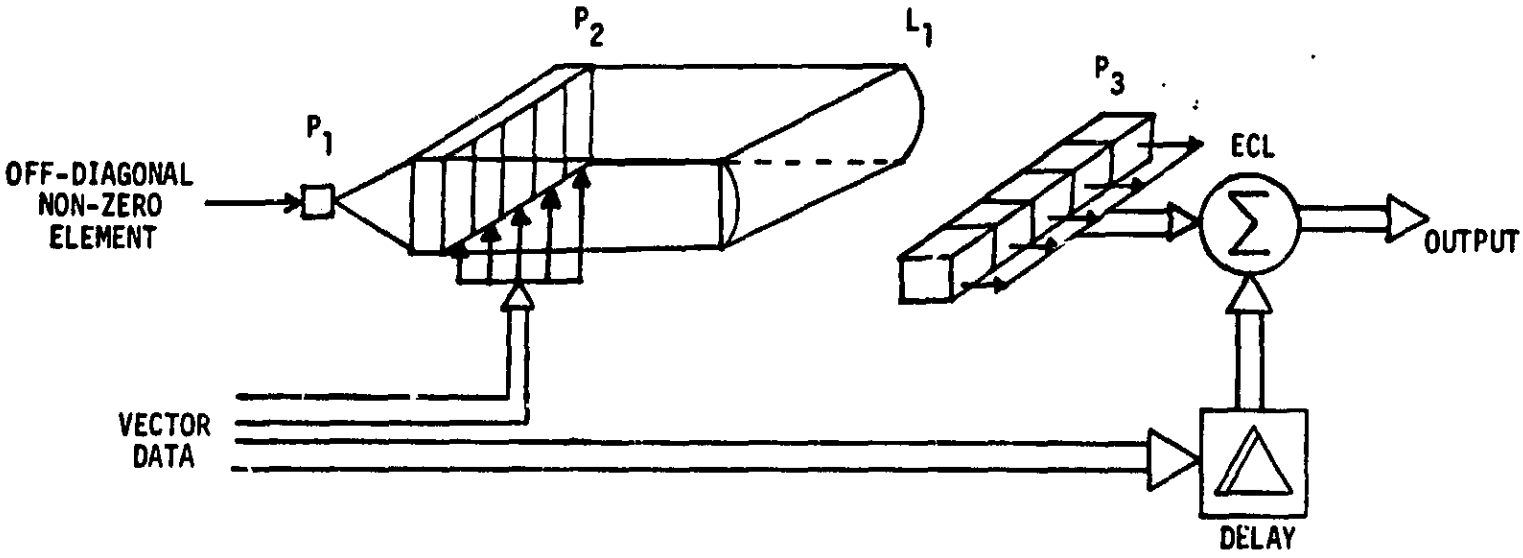


Figure 4-12: One-Channel Finite Element Processor

The P_1 point modulator is used for the input of the non-zero data that is not from the main diagonal of P^k which is always a 1. The multi-channel AO cell at P_2 need be only a point modulator array. Its input is fed to a delay, and the output from the detector array in P_3 and the delayed P_2 data is added in a

serial ECL circuit at the appropriate T_1, as if it had been multiplied by 1 in the processor. In practice the ECL adder in the original detector circuit (Figure 4-8) is used. If the 12^{th} bit was the one's place in the 32-bit representation being used, the P_2 data would be delayed 11 T_1's, and added to the detector output of the 12^{th} T_1, within each T_2 period. This one-channel implementation performs the same multiplications indicated in Figures 4-9 and 4-11, for the LU decomposition algorithm.

This processing format allows bipolar handling with an exclusive-or test, as detailed in section 4.3, since only one input channel is used. If the exclusive-or test result is a 1, indicating that the signs of the data in P_1 and P_2 are different, the product is given a negative sign. In this case, the delayed data is subtracted rather than added to the detector output at the proper time.

The dynamic range requirement of the detector is now drastically reduced, from that discussed in section 4.4. Using a 1-channel processor as described above, or the M=10 channel processor, with the LU decomposition algorithm, only one non-zero input will be present in either system every T_1. For binary encoding, each detector element (for the detector array of Figure 4-8) requires a dynamic range of 1. Each A/D converter must convert two levels, a 0 or a 1. Thus, one bit A/D converters are required, and such units (simple comparator circuits) are available at speeds above 100 Mhz.

If the data encoding is performed in a radix R, each detector would require a dynamic range of $(R-1)^2$. Each A/D converter would require $\log_2[(R-1)^2+1]$ bits, rounded to the next largest integer. If the data were encoded in a radix R other than two (binary), the dynamic range requirements of the detector system are larger than the R=2 case. However, fewer bits are required for the encoding to achieve the same accuracy. Thus, the number of channels in P_2 is reduced, and the speed of the processing is increased, since there will be fewer T_1's per T_2.

Two more points should be made about our proposed finite element processing system. When implementing the LU decomposition algorithm on the processor as described, no frequency multiplexing is used in the N-channel cell in P_2. If frequency multiplexing is used in an optical system, the number of frequencies that can be used is limited by the divergence of the input beam, as explained in [25]. Thus, beam divergence is not an important consideration in fabrication of the input array for our processor. Also, the proposed processor is basically an imaging system. No information is carried in the phase of the light distributions. The multiplications are represented by the beam's intensities, thus non-coherent light can be used in the processor.

4.7 Solving Large Systems of Equations

This section will address two issues, large banded matrix problems in general, and large finite element problems. The first issue arises from the practical limitation of the size of the input point modulator array. An array of M=10 point modulators was chosen because it is a reasonable number for fabrication. On a larger scale, fabrication of a processor with more than 50 channels looks unrealistic, especially because the time aperture of the AO cell in P_2 would have to be very long to facilitate reasonable shift rates in P_3. The second issue involves the standard finite element problem formulation technique of substructuring, which enables large problems to be processed in a more efficient manner.

4.7.1 Matrix Partitioning

The previous section showed that any size banded matrix problem, solved with the LU decomposition algorithm implementation on our processor, only requires a one channel system, shown in Figure 4-12. This special requirement is due solely to the nature of our implementation of the LU decompositon algorithm. It is important to consider the more general purpose uses of our M=10 channel processor. As mentioned previously, many other algorithms are suitable for implementation on the processor, especially those that involve banded matrix-vector

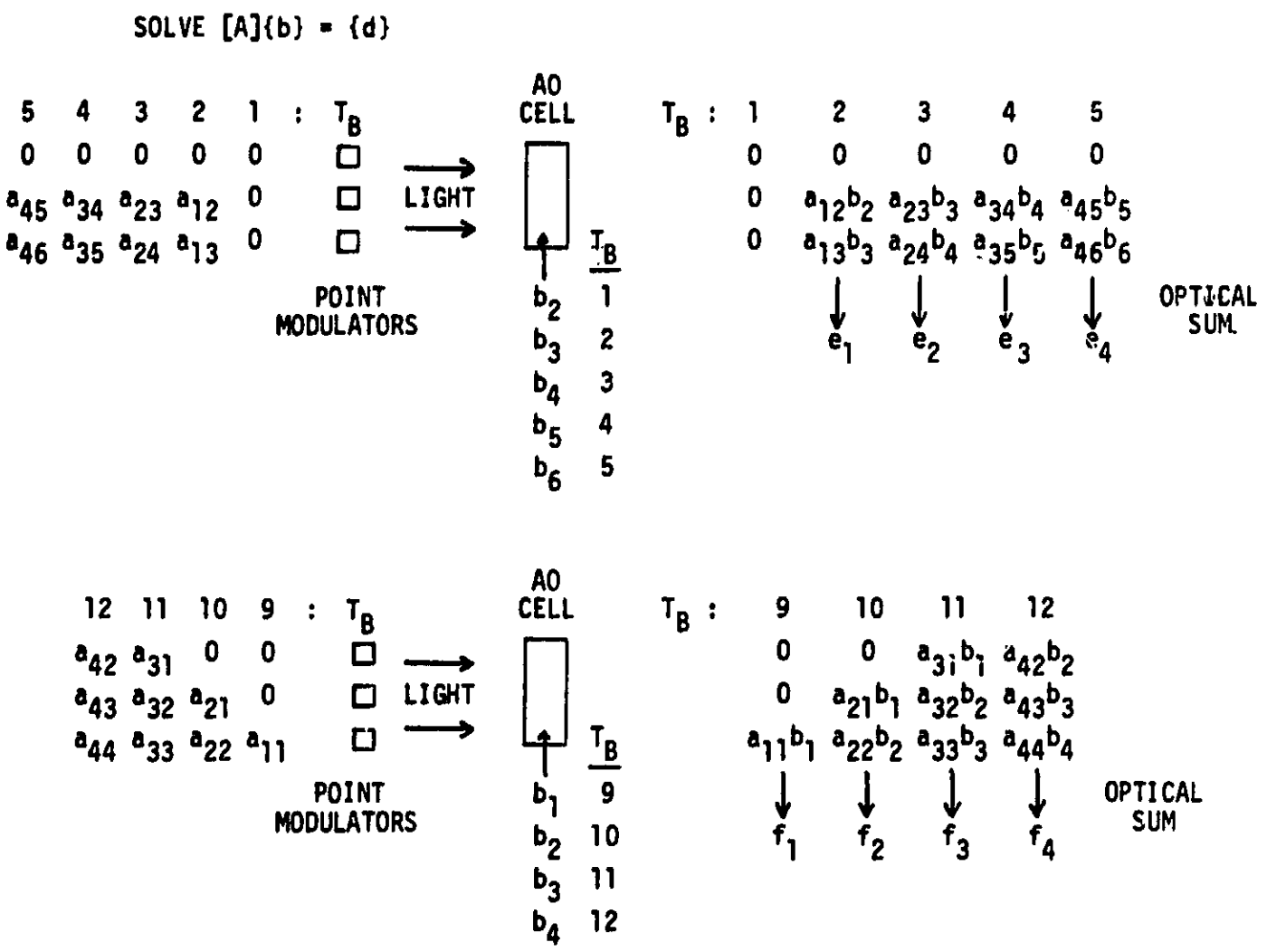
products, which is a basic operation defined in Figure 4-9. They include other direct equation solvers, iterative equation solvers, and other algorithms.

By examining Figure 4-9, it is obvious that our M=10 channel processor is limited to processing matrix equations with a bandwidth of 10 or less, when performing a banded matrix-vector product. However, most significant problems will have bandwidths that are much larger than 10. Finite element problems often have tens or hundreds of thousands DOFs, and bandwidths in the thousands or tens of thousands. These large problems cannot be solved on the M=10 channel processor without some form of partitioning of the equations, to reduce the amount of data that must be fed to the processor at one time. In considering partitioning schemes, two issues must be kept in mind: 1) a system for solving banded matrix equations is being used, and any partitioning must be compatible with that format; 2) the partitioning must not be so complex that is slows the processing time considerably.

An appropriate partitioning scheme is now discussed. Figure 4-9 illustrates how a banded matrix-vector product is implemented on our optical processor. The number of point modulators required is equal to the bandwidth of the matrix. Figure 4-13 illustrates how a banded matrix-vector product can be implemented when the matrix bandwidth B exceeds the number of point modulators M in the processor. The matrix is partitioned such that M (the number of point modulators) or less diagonals are processed at any one time. Recall that in our implementation, an entire diagonal is fed into a single point modulator, thus making this partitioning scheme attractive. The detector results are stored and added accordingly to yield the desired vector elements. The matrix of Figure 4-9, with a bandwidth of 5, is partitioned for the processor of Figure 4-13, with only M=3 point modulators.

The top two diagonals of the matrix equation in Figure 4-9 are fed into the processor first, as illustrated in the top portion of Figure 4-13. Note that one of

Figure 4-13: Banded Matrix-Vector Product With Matrix Partitioning



$$d_i = e_i + f_i, \quad i = 1, \cdots, 7$$

the inputs is not used. This causes no data flow problems when the unused inputs are positioned at the end of the AO cell. The partial products produced from the top two diagonals are optically summed to produce 7 e_i values (only the first 4 are shown in the figure). The remaining three diagonals of the matrix equation are then fed to the processor as illustrated in the lower portion of Figure 4-13. The partial products from that operation are optically summed to produce 7 f_i values. The sum of e_i and f_i yields the desired element d_i (from Figure 4-9) of the vector d. This can be verified by comparing the results of Figure 4-13 with those in Figure 4-9.

It is important to note that with this partitioning scheme, the data flow is nearly always constant through the processor. In the example of Figure 4-13, 7 bit times, T_B, are needed to produce the e_i's. Actually, e_7 is zero and not produced since no element of the top two diagonals contributes to the formation of the vector element d_7. The e_6 value is produced at bit time 7, when b_7 is in the middle region of the cell. Bit time 8 is used to shift b_7 to the top region of the cell, and at bit time 9, processing of the remaining three diagonals begins.

The LU decomposition algorithm can be implemented on the M=10 channel processor for large problems with this partitioning scheme. The diagonals of P^k are partitioned and processed in groups of 10 or less, as illustrated in Figure 4-13. The results from each partition are stored in the microprocessor, and they are added appropriately to form the desired matrix-vector products. If a partition includes the main diagonal of P^k, the processor has at most two non-zero point modulator inputs. If the partition does not include the main diagonal, the processor has only one non-zero point modulator input. Thus, if the main diagonal is partitioned alone, the processing will take place with only one of the M=10 point modulators input with non-zero data at any time. In this case, bipolar data can be handled with the exclusive-or test, since it is known which point modulator will be fed the non-zero data during each T_2.

4.7.2 Substructuring

The large size of many finite element problems, having tens of thousands of DOFs, presents a data management and formulation problem for any computing system, not to mention the large amount of CPU time needed for the problem solution. A technique called substructuring has been used to overcome these problems. Structures often have well-defined or definable internal boundaries that can be used to separate the structure into a system of substructures. The boundaries between substructures exist because of symmetry, construction, or other constraints. Each substructure may possess similar internal boundaries, and can be further subdivided into other substructures. Thus, a complex structure can often be reduced to a connection of smaller and simpler substructures. This breakdown simplifies the structure modelling. The following discussion will deal with the technique of substructuring with condensation. Substructuring without condensation is also a useful technique in problem formulation, but it will not be discussed here.

The equations for a finite element model made up of substructures (formulated by condensing internal nodes, to be discussed below) can contain many fewer DOFs than the equations for the model without substructures. This significantly reduces the solution time and the amount of data to be handled in the problem. Defining the problem in terms of substructures reduces the problem formulation to one with many small manageable matrix equations. The benefits of breaking up large problems into smaller substructures are obvious, and the process is detailed below. Substructuring is not benificial for all problems, especially smaller ones, but when used properly, it can save significant amounts of problem formulation and solution time.

Most substructuring exploits repeated geometries and symmetry within a structure. A trivial illustrative example of one level of substructuring is given for the simple structure of Figure 4-14. The structure is modelled by 24 triangular elements, and many lines of symmetry exist. Each half of the structure about the vertical line of symmetry can be considered to be a substructure. Both of these

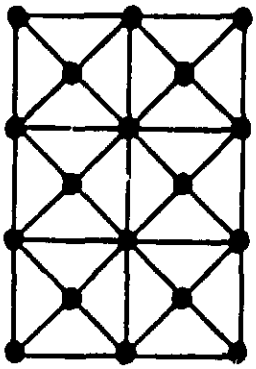


Figure 4-14: Example Model Suitable for Substructuring

substructures are identical. Once the proper substructure equations have been derived (through condensation), the structure may be modelled as shown in Figure 4-15.

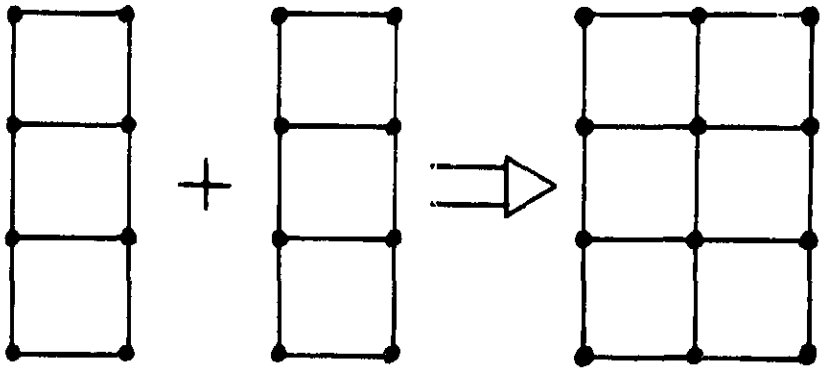


Figure 4-15: Substructured Model

Here, two identical substructures, rather than 24 elements, model the structure. Interior nodes have also been condensed in Figure 4-15 as detailed later. Since the substructures have identical geometries, the same set of equations will describe both of them. Thus, repetition of input data, storage, and calculations is eliminated. The problem size is also reduced, from 18 nodes in Figure 4-14 to 12 nodes in Figure 4-15. The substructures (combinations of many elements described by a set of equations derived from those elements) may be thought of as superelements.

The substructure equations are derived through a procedure called static

condensation. In general, for a given substructure or element, exterior nodes are those nodes needed to connect the substructure or element to other substructures or elements, i.e. those nodes on the boundaries. All other nodes are interior nodes. Static condensation is used to condense the interior nodes by expressing their equations in terms of the exterior nodes, thereby expressing the entire substructure in terms of the exterior nodes only. In Figure 4-15, the interior nodes have been condensed, and thus only the exterior nodes of each substructure are shown.

As a simple example of static condensation, consider the structure composed of two bar elements in Figure 4-16.

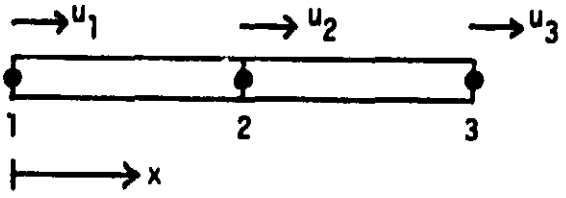


Figure 4-16: Static Condensation Example - Bar Element Model

The structure has one DOF per node i, specifically a displacement u_i in the x direction. The basic finite element matrix equation for this problem is

$$\begin{bmatrix} k_{11} & k_{12} & 0 \\ k_{21} & k_{22} & k_{23} \\ 0 & k_{32} & k_{33} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \tag{4.22}$$

To condense interior node 2, the equation from the second row of the matix in (4.22) is used to write u_2 in terms of u_1 and u_3, as

$$u_2 = (p_2/k_{22}) - (k_{21}/k_{22})u_1 - (k_{23}/k_{22})u_3 \tag{4.23}$$

Equation 4.23 is substituted into the equations from the first and third rows of the matrix in (4.22) to yield the equations

$$\begin{aligned} [k_{11}-(k_{12}k_{21}/k_{22})]u_1 - (k_{12}k_{23}/k_{22})u_3 &= p_1 - (k_{12}/k_{22})p_2 \\ -(k_{32}k_{21}/k_{22})u_1 + [k_{33}-(k_{32}k_{23}/k_{22})]u_3 &= p_3 - (k_{32}/k_{22})p_2 \end{aligned} \tag{4.24}$$

The equations in (4.24) can be written in matrix form as

$$\begin{bmatrix} k_{11}-(k_{12}k_{21}/k_{22}) & -(k_{12}k_{23}/k_{22}) \\ -(k_{32}k_{21}/k_{22}) & k_{33}-(k_{32}k_{23}/k_{22}) \end{bmatrix} \begin{bmatrix} u_1 \\ u_3 \end{bmatrix} = \begin{bmatrix} p_1-(k_{12}/k_{22})p_2 \\ p_3-(k_{32}/k_{22})p_2 \end{bmatrix} \quad (4.25)$$

Equation 4.25 is a reduced dimensionality form of (4.22), and can be used to represent the structure of Figure 4-16 (as long as no connections are made at node 2) in terms of nodes 1 and 3. Equation 4.25 thus represents the substructure in Figure 4-17.

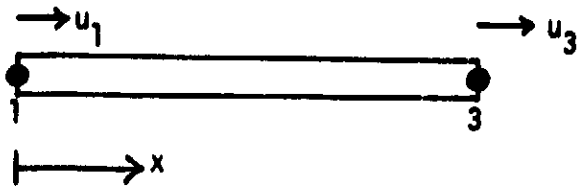


Figure 4-17: Bar Example Substructure - Node 2 Condensed

It may be considered as a superelement, since (4.25) has the same form as a set of elemental equations (the matrix is full, in general). Since it is used in a structure equation assembly process just like a single element, the assembly of substructure equations will yield a banded matrix as before, if the substructure nodes are numbered properly.

When condensed equations for substructures are used in the assembly process, smaller structure stiffness matrices result. They are smaller by the total number of condensed DOFs in the structure (if a condensed node has n DOFs, n equations are eliminated). If a substructure is repeated throughout the main structure, savings in input, storage and calculations result since the substructure equations need only be derived once. The same set of equations is used over and over in assembling the repeated substructures into the structure stiffness equations. This situation is synonomous to remarks made in Chapter 2 that elemental stiffness matrices need only be evaluated once for like elements, and they are used more than once in the stiffness matrix assembly.

The condensation process can be described in general by the following equations. The matrix stiffness equation for the substructure to be condensed is written as

$$\begin{bmatrix} K_{ii} & K_{ie} \\ K_{ei} & K_{ee} \end{bmatrix} \begin{bmatrix} u_i \\ u_e \end{bmatrix} = \begin{bmatrix} p_i \\ p_e \end{bmatrix} \tag{4.26}$$

where u_i is a vector of the (interior) DOFs to be condensed, and u_e is a vector of the (exterior) DOFs to be retained. The load vector and stiffness matrices are also partitioned in terms of the DOFs to be condensed and retained. A condensed matrix equation is desired of the form

$$K_{ee}' u_e = p_e' \tag{4.27}$$

After simple matrix manipulations of the partitioned equations in (4.26), the following relations are obtained

$$\begin{aligned} K_{ee}' &= K_{ee} - (K_{ei} K_{ii}^{-1} K_{ie}) \\ p_e' &= p_e - (K_{ei} K_{ii}^{-1} p_i) \end{aligned} \tag{4.28}$$

The equations in (4.28) can be verified by comparing them to (4.25).

The condensed substructure matrix equation can be obtained from (4.28), however, in practice this is not done because of the matrix inversion that is required. Instead, modified forms of standard equation solvers are applied to (4.26), such as Cholesky or Crout decomposition, or Gaussian elimination. For example, Gaussian elimination can be applied to (4.26) until all the elements of K_{ei} (the lower partition of (4.26)) are zero, yielding (4.27).

As mentioned earlier, substructures can be defined within substructures, and thus many levels of substructuring are possible. A three-level example follows, taken from [33]. The main structure is shown in Figure 4-18. It is a three-bay, three-story steel frame; the second bay girders are identical, each having three openings (penetrations) for the passage of ductwork (Figure 4-19a). Most of the

structure can be modelled with simple plane frame elements. However, the three penetrated girders require many nodes and elements for proper modelling of the openings Substructuring and condensation can be used to exploit the repeated geometries and symmetry in each penetrated girder. This will greatly simplify the entire structure model.

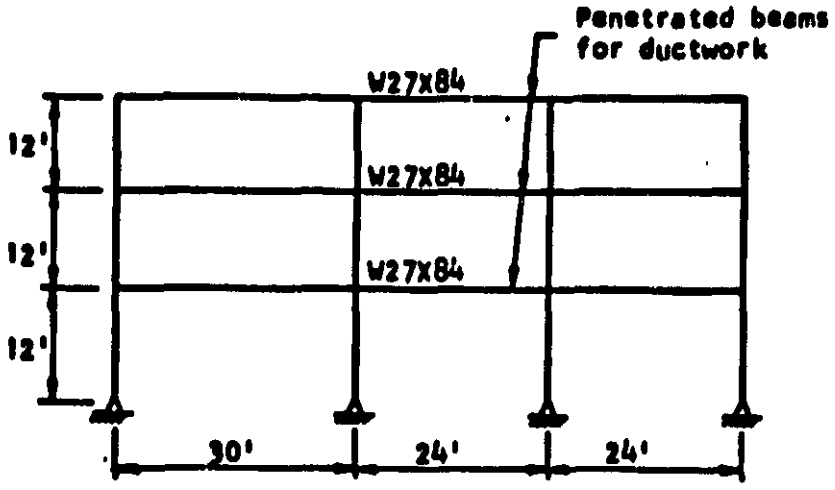


Figure 4-18: Building Frame - Bays Horizontal, Stories Vertical [33]

The first level substructure is a single penetrated girder (Figure 4-19a). The central (second) bay of the entire stucture is modelled with three copies of Figure 4-19a, with plane frame elements used for the beams and columns in the first and third bays. The second level substructure (Figure 4-19b) is defined by dividing Figure 4-19a into three identical parts. The third level substructure (Figure 4-19c) is defined by subdividing Figure 4-19b into four identical parts, each with a different orientation (rotation transformations will be required). The modelling of the

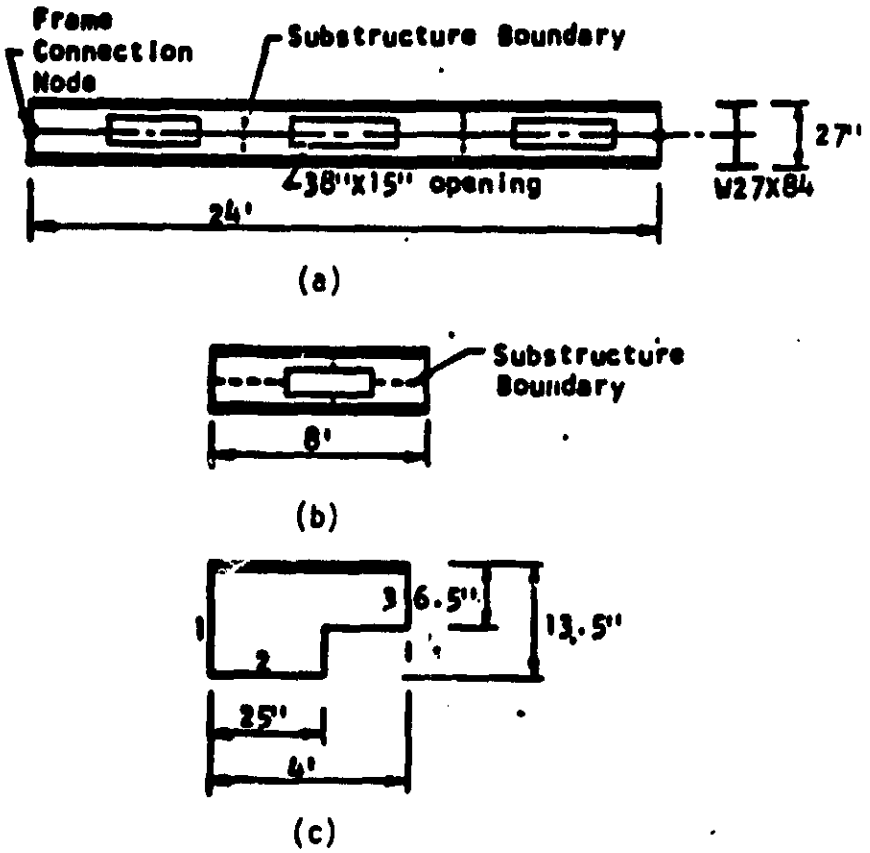
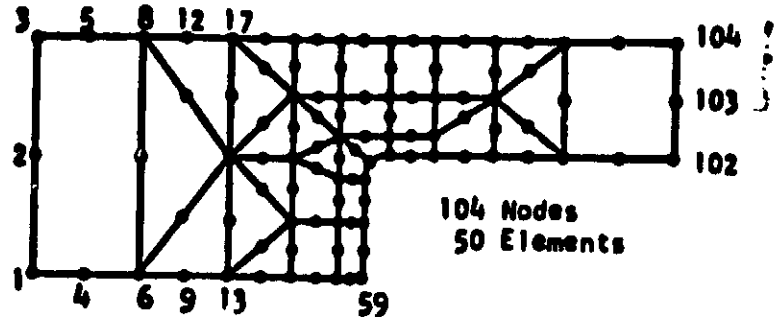


Figure 4-19: Penetrated Girder Substructures [33]

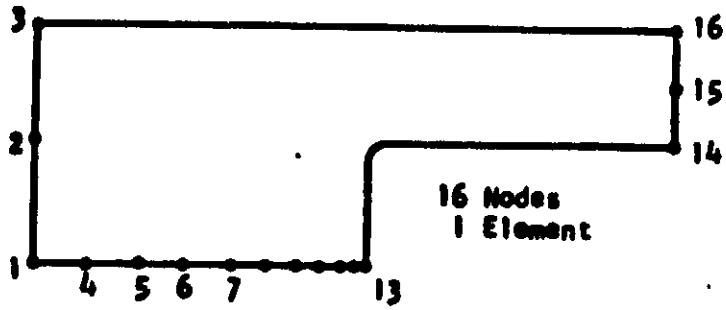
third level substructure is shown in Figure 4-20a. It consists of standard finite elements (rods, triangles, and rectangles) as shown.

All the interior nodes of Figure 4-20a can be condensed, to yield the substructure, or superelement in Figure 4-20b. Four of those are used to form the second level substructure, and three such second level substructures are used to form the first level substructure of Figure 4-20c. The interior nodes of Figure 4-20c are then condensed to yield the first level super element (substructure) shown in Figure 4-20d, which has only two nodes.

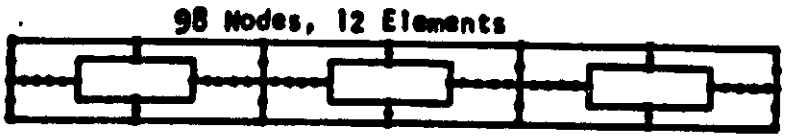
Figure 4-20. Building Frame Model Substructures [33]



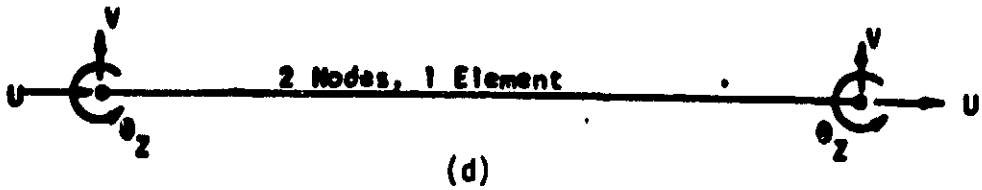
(a)



(b)



(c)



(d)

Three copies of Figure 4-20d, along with simple plane frame elements, are used to model the original structure of Figure 4-18. The number of DOFs in this model is much less than if substructuring was not used (i.e., if all the nodes of Figure 4-20a were modelled explicitly). Each first level substructure contains 1152 condensed nodes. Since the first level substructure is defined by 12 identical third level substuctures, duplicate input data, computation, storage of elemental stiffness matrices, etc., is eliminated from the problem formulation. The final matrix equation to be solved is much smaller, thus the solution process will be much faster.

When a finite element problem is solved using substructuring and condensation, the condensed DOF values are not determined. However, if they are needed for the analysis, they are retrievable by back-condensation. This process is the reversal of the steps used to obtain (4.27), which are known.

The main advantage of substructuring with condensation is that the size of the structure finite element equations is greatly reduced due to the condensed DOFs. This significantly reduces the required computer time for solving problems on a digital system. For processing on our optical system, it reduces the matrix and bandwidth size, so that the equations may be more easily managed, and that less partitioning is required for algorithms other than our LU decomposition implementation.

If substructuring is used when inappropriate, the overhead required for the substructuring operations will exceed any gains in handling and solving the equations after substructuring. Five general guidelines for substructuring are given in [33], and are summarized below:

1. A condensed substructure should be used more than once as a superelement in a higher level structure.
2. The number of nodes remaining after condensation should be small compared to the total number of substructure nodes.
3. Nodes remaining after condensation should form a narrow boundary within the structure, which minimizes the structure bandwidth.

4. Nodes should be numbered to exploit any node reordering scheme used in the condensation algorithm.

5. The number of nodes requiring back-condensation should be significantly less than the total number of condensed nodes in the final model.

4.8 Summary and Conclusion

This chapter has presented an optical processing system suitable for accurately solving finite element problems, and other banded systems of equations. The limitations of analog optical processors were discussed, and some proposed digital encoding schemes were reviewed. Multiplication by digital convolution was explained and detailed for an optical system. A specific digitally encoded optical processor was described, which performs digital convolution by the standard shift-and-add method. Its performance was evaluated, and fabrication of a realizable system was discussed.

It was shown how banded matrix-vector problems could be performed on the proposed optical processor. An LU decomposition algorithm was detailed to solve finite element matrix equations. The algorithm uses banded matrices, and thus is appropriate for our processor. It was shown how our implementation of the LU decomposition algorithm actually requires only one processor channel. Matrix partitioning for other algorithms was detailed. The topic of substructuring in finite element problem formulation was described, which can often be used to make the formulation, handling, and solution of finite element equations much easier.

This chapter has presented a feasible optical processing system and associated algorithms. The digital encoding used by the processor will yield accurate solutions to finite element problems. This was obtained by trading off speed and size of the processor for the encoding capability. This chapter is unique in recent optical processing literature in that it described existing hardware that can be employed to input and output data to and from the processor. It remains to be seen how the errors inherent in optical systems will affect the performance of this digitally encoded processor. This is the subject of the Chapter 5 and future research.

5. OLAP Simulation

5.1 Introduction

Chapter 4 has described an optical processor capable of performing basic linear algebra operations. Thus, we will now refer to our proposed processor as an optical linear algebra processor, or OLAP. The OLAP employs digital encoding of data to reduce the dynamic range of the signals in the processor and lessen the effect of optical processor errors on the data. It is expected that significant optical errors can be present in the OLAP without significantly affecting the processing accuracy. This chapter describes a digital computer simulation of the OLAP used to solve three selected multiplications of two binary encoded numbers. The purpose of the simulations is to determine how optical errors affect the products calculated in a digitally encoded optical processor.

The OLAP error simulation program was written to simulate the solution of a case study finite element problem with individual and combined optical error sources present, and it is fully detailed in this chapter. However, this report only includes the simulation results for three sets of multiplications, with only individual error sources included in each simulation. This is because a single simulation run for our case study with N=24 (after imposing boundary conditions) requires 4 hours of CPU time, as will be explained in section 5.3. A comprehensive processor evaluation will require nearly 100 simulation runs (due to trial and error, and individual and combined error source modelling), which is over two weeks of CPU time. Thus, that task will be completed in future research.

All of the pertinent error sources present in the OLAP are modelled in the simulation program. These error sources are described in section 5.2. The direct LU decomposition algorithm detailed in Chapter 4 is used. The simulation program models the full implementation of this algorithm on the M=10 channel OLAP of Figure 4-6, with N=32 bits. We chose to simulate the 10-channel system rather than the 1-channel system (section 4.6) because it represents a more flexible

general-purpose processor. This is an appropriate choice for two reasons. First, the specific algorithm (LU decomposition) and implementation we chose to solve finite element problems lent itself to a 1-channel OLAP, but other algorithms may not do this. Second, the error simulation results are more significant if they are for a more general processor, with multiple processor channels.

The error modelling assumptions and software implementation are discussed (section 5.3). The results of the individual 32-bit multiplication simulations are then advanced (section 5.4), with analysis of the results also advanced. The simulations were performed on a VAX 11/750, with single precision (32-bit floating point) arithmetic. This chapter thus presents the first model and quantitative simulation results on the performance of optical processors operating on digital data in the presence of various component errors.

5.2 Error Sources and Error Modelling

This section describes the error sources considered, and the modelling used for the OLAP of Figure 4-6. This represents the first error source model and analysis for a digital OLAP. For the error model, M=10 channels and N=32 bits are used, and the detector array used at P_3 is the detector/ECL array shown in Figure 4-8, and discussed in section 4.4.

The processor is repeated for convenience in Figure 5-1. Only the top and bottom portions of the P_1 and P_2 devices are shown. There are 10 P_1 input point modulators, 32 AO cell channels with 10 T_2 regions present in P_2, and 32 detector elements are used in P_3. This figure will be used to describe our error source modelling. The input point modulators are numbered 1 to 10 and are indexed by i. The signal input to point modulator i is described by a_i. The AO cell channels in P_2 are numbered 1 to 32 and are indexed by j. Each channel is divided into 10 regions corresponding to the 10 input point modulators. The ideal P_2 plane transmittance of AO cell channel j at region i is described by b_{ij}. The detector elements are numbered 1 to 32 and are also indexed by j. The

value incident on a detector element is described by c_i. As defined in Chapter 4, processor channel i consists of input point modulator i in P_1, and the region i of P_2 containing all j=1 to 32 AO cell channels.

The error sources modelled are those that are most dominant. Many error sources could be included in a digital model of an optical processor. However, these error sources would depend on the specific processor hardware, and their number would become prohibitive for a simulation. The more reasonable and useful approach is taken in which the major dominant error sources are lumped into combined errors in each data plane, thus greatly simplifying the simulator and its runtime. The effect of the various error sources is thus more easily quantifiable since there are fewer error sources. Such a simulation enables an objective determination of the effect of combinations of errors in various planes (and of various types) on the OLAP's performance.

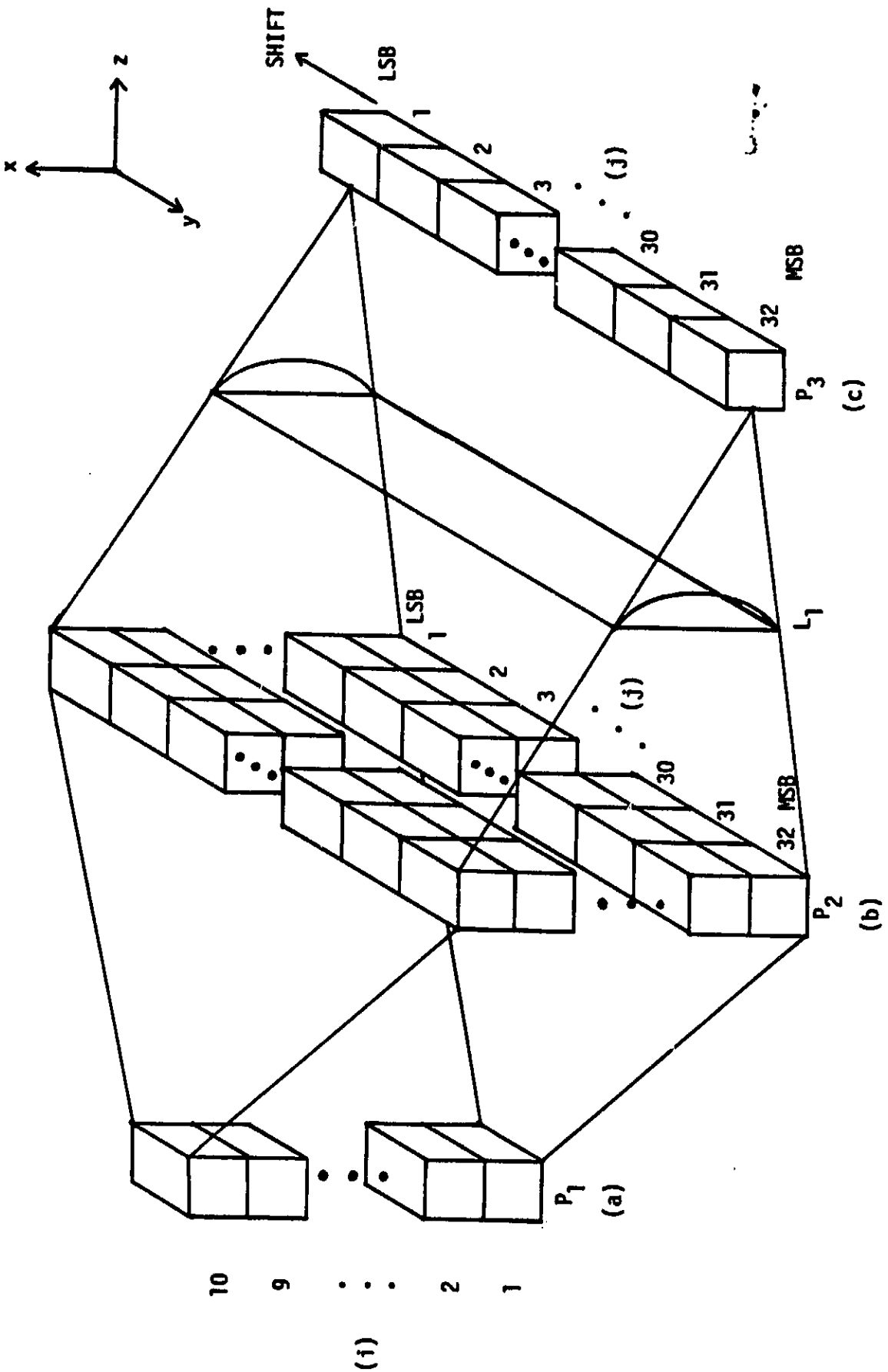
The error source model is separated into errors in the three planes of Figure 5-1: the input plane, P_1; the multi-channel AO cell plane, P_2; and the detector plane, P_3. Seven error sources are modelled as indicated in Table 5.1.

5.2.1 Input Plane P_1 Errors

The single input plane P_1 error is spatial gain. This term refers to the gain differences between the input point modulators, i.e., the differences across the spatial dimension of the input plane P_1. The ideal transfer function for each point modulator, converting input volts to output light intensity, is shown in Figure 5-2. The ideal transfer function has a slope g, the gain of the point modulator, which can be scaled to one. For a real point modulator, the gain will not be exactly one, but can be adjusted to a value close to one. However, some residual error will exist. This error changes the slope of the transfer function plot, and is thus multiplicative in nature (introducing a small error in small inputs, and a large error in large inputs).

We write the input to point modulator i, as a_i, and describe the light intensity leaving point modulator i by

Figure 5-1: Optical Linear Algebra Processor



Error Source	Notation
Input Plane Errors:	
Spatial Gain	$1+\delta_i^1$
Multi-Ch. AO Cell Plane Errors:	
Acoustic Attenuation	$\exp(-\alpha x_i)$
Adjacent Ch. Crosstalk	--
Even-Odd Ch. Crosstalk	--
Detector Plane Errors:	
Spatial Response	$1+\delta_j^3$
Time-varying Noise	$n_j(t)$
Dark Current	d_j

Table 5-1: Error Sources for OLAP Error Source Model

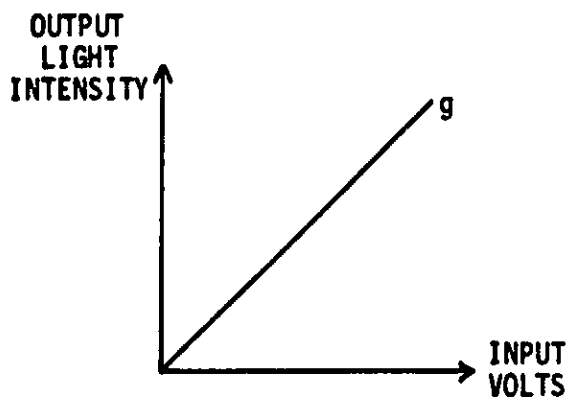


Figure 5-2: Ideal Point Modulator Transfer Function

$$\hat{a}_i = a_i(1+\delta_i^1) \tag{5.1}$$

The quantity δ_i^1 models the residual gain error (which modifies the ideal gain of unity), and thus $(1+\delta_i^1)$ multiplies the input value and models the multiplicative P_1 spatial gain error.

Two other input plane P_1 errors can be included in the error source modelling of optical processors [35]. The first is a nonuniform response for the P_1 modulators. This represents a bias level in the transfer function, i.e., the plot

in Figure 5-2 does not pass through the origin, and the bias is different for each point modulator. This error source can be modelled as a simple additive spatial error added to each input. The other input P_1 error is spatial coupling, which represents the coupling, or crosstalk between inputs on different input point modulators. These latter two error sources are not included in our initial simulations.

5.2.2 Multi-Channel AO Cell Plane P_2 Errors

The first multi-channel AO cell plane P_2 error considered is acoustic attenuation. The signal presented to the transducer of each AO cell channel attenuates exponentially as it moves across the cell in the x direction in Figure 5-1. The N=32 signals in region 1 of P_2 are at full strength. As the signals move to each successive T_2 regions, their strength is decreased by an exponential factor, and are weakest when at the last or end region 10. Acoustic attenuation depends on the frequency and AO cell material. For our system, it is the same for all channels in the multi-channel P_2 AO cell (since only one frequency is used).

We model the effect of acoustic attenuation by multiplying the signal at the transducer by the exponential factor $\exp(-\alpha x_i)$. The constant α is the attenuation constant determined by the AO cell material and operating conditions. It is expressed in dB/mm and is converted to Nepers/mm for use in our exponential model, where 1 dB equals 0.23 Nepers. The x_i value is the location in mm corresponding to P_1 point modulator i and region i of the P_2 AO cell. In our simulation, region 1 is considered to have no attenuation, thus x_1 is zero. If the point modulators are centered 2 mm apart, then x_2 is 2 mm, x_3 is 4 mm, etc. Including only acoustic attenuation errors in P_2, the transmittance of region i of AO cell channel j is written as

$$\hat{b}_{ij} = b_{ij}\exp(-\alpha x_i) \tag{5.2}$$

where b_{ij} is the unattenuated signal input to channel j at time $-iT_2$.

The second P_2 error source is crosstalk between adjacent AO cell channels. Specifically, the signal input to AO cell channel j affects the signals in AO cell channels j-1 and j+1. We express this error in dB. If a signal of level I_1 in channel j induces a signal of level I_2 in channels j-1 and j+1, then there is X dB of isolation between these adjacent channels, where

$$10LOG_{10}(I_1/I_2) = X \tag{5.3}$$

The third P_2 error source is even-odd channel crosstalk. This error source was included because this type of crosstalk was found to be significant in preliminary tests of the 10-channel AO cell. It is modelled in the same way as adjacent channel crosstalk, except that a signal in channel j induces crosstalk in channels j-2 and j+2.

Modelling crosstalk effects is slightly subjective since it is hard to distinguish between RF crosstalk at the transducers and acoustic wave crosstalk when making measurements. It is obvious, however, that the even-odd channel crosstalk is entirely due to RF sources. Crosstalk effects are additive, and the following terms are added to (5.2) to model both types of crosstalk:

$$ADJ = 1/10[\sum_{i=1}^{10}(b_{i,j-1}+b_{i,j+1})][I_2/I_1]$$

$$EO = 1/10[\sum_{i=1}^{10}(b_{i,j-2}+b_{i,j+2})][I_2/I_1] \tag{5.4}$$

The term ADJ is the contribution due to adjacent channel crosstalk, and EO is the contribution of even-odd channel crosstalk. The 1/10 factor is included to distribute the crosstalk equally over all 10 regions of an AO cell channel. The I_2 and I_1 values (the relative signal levels in two channels, I_2 due to crosstalk from I_1) are those defined in (5.3), and may differ in the two expressions in (5.4).

Spatial gain errors and similar effects could have been modelled for the multi-channel AO cell as a P_2 error. However, these errors can be lumped into

the spatial gain error in P_1, or the spatial response error in P_3, mentioned below. Frequency response errors in P_2 were not included in our initial error model (i.e. we assume a flat frequency response), but can be included later if frequency multiplexing is used. This error can be modelled as a P_2 or P_3 (the Fourier Transform plane) error.

5.2.3 Detector Plane P_3 Errors

The first detector plane error considered is spatial response. This term refers to the response, or gain, differences between detectors in the P_3 array in space. This error source is modelled the same as the spatial gain error for the input P_1 array. The ideal transfer function for each detector element, converting input light intensity to output current or voltage, has the form in Figure 5-2, and this error is also multiplicative. The second P_3 error source included is time-varying noise. This error source is a result of shot noise and thermal noise (due to the finite temperature of the detector system). We assume that this noise is not signal dependent [36]. This error is additive because the noise is random and always is present on the detector, regardless of the intensity of light incident on it. The final P_3 error source included in our model is detector dark current. This current is a result of thermal electron-hole pair generation in solid-state devices. It is basically a constant background current, but differs for each detector element and it thus another additive error source.

We write the observed detector output, c_j, in terms of the three P_3 errors defined above, as

$$\hat{c}_j = c_j(1+\delta_j^3) + n_j(t) + d_j \qquad (5.5)$$

where c_j is the exact signal incident on detector element j, δ_j^3 is the residual error from the ideal response, $n_j(t)$ is the time-varying noise, and d_j is the dark current.

5.2.4 Combined Error Source Model

A closed form expression can be written to describe the 32 observed detector outputs at one T_1 in terms of the inputs to the processor, and the error sources in Table 5.1 (except for the two crosstalk errors, which do not fit easily into the closed form expression) as

$$\begin{bmatrix} \hat{c}_1 \\ \hat{c}_2 \\ \vdots \\ \hat{c}_{32} \end{bmatrix} = \begin{bmatrix} 1+\delta_1^3 & & \\ & 1+\delta_2^3 & \\ & & \ddots \\ & & 1+\delta_{32}^3 \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{2,1} & \cdots & b_{10,1} \\ b_{1,2} & & & \\ \vdots & & & \\ b_{1,32} & & \cdots & b_{10,32} \end{bmatrix} \begin{bmatrix} 1+\delta_1^1 & & \\ & \ddots & \\ & & 1+\delta_{10}^1 \end{bmatrix} \begin{bmatrix} e^{-\alpha x_1} & & \\ & \ddots & \\ & & e^{-\alpha x_{10}} \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_{10} \end{bmatrix} + \begin{bmatrix} n_1(t) \\ n_2(t) \\ \vdots \\ n_{32}(t) \end{bmatrix} + \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{32} \end{bmatrix}$$

32×1 OBSERVED P_3 DATA	32×32 DET. SPAT. RESPONSE	32×10 P_2 DATA	10×10 INPUT SPAT. GAIN	10×10 ACOUSTIC ATTN.	10×1 P_1 DATA	32×1 TIME-VARYING NOISE	32×1 DARK CURRENT

(5.6)

where all of the notation has been defined previously. Superscripts 1 and 3 refer to the planes P_1 and P_3, respectively. The P_2 data matrix is actually an encoded vector, thus requiring two dimensions rather than one. Note that the subscripts in the P_2 data matrix are transposed with respect to conventional row-column matrix subscripts. This is simply due to the data indexing used in Figure 5-1. Since each row in (5.6) represents one detector output, each row of the P_2 data matrix contains the data in all 10 T_2 regions (the first subscript) of each AO cell channel, indicated by the second subscript.

It is necessary to model the various error sources such that we can quantify the degree to which they must be reduced (that is, the allowed levels of residual

spatial errors) for successful processor operation. Specifically, we wish to quantify the allowed levels of residual spatial errors, acoustic attenuation, time-varying noise, and detector dark current that can be allowed in the processor without creating significant error in the processing results. Our error source simulator is also intended to determine the dominant error sources and how combinations of several error sources (present simultaneously) interact and combine to affect performance. In our processor, we distinguish between correctable errors (residual errors) and the uncorrectable errors ($n_j(t)$ and crosstalk). Spatial P_1 and P_3 errors can be corrected to levels set by the system's measurement accuracy (typically 0.1%). Spatial P_2 errors, (e.g., acoustic attenuation) can likewise be corrected, for one frequency operation.

5.2.5 Simulation of Error Sources

We now discuss how the various errors are modelled in our simulator. The distribution of spatial errors across the input and detector arrays can best be described by a zero-mean Gaussian distribution. Since the input gain and detector response corrections are made with finite accuracy, the residual errors are random. Such random events typically follow a Gaussian distribution. Reference [37] verified that residual spatial nonuniformities are indeed Gaussian for an optical processing system. The modelling used for these errors is described below.

A multiplicative error applied to a quantity y yields $\hat{y}$, where

$$\hat{y} = y(1+\sigma_1 D) \tag{5.7}$$

and $\sigma_1 D$ is a random multiplicative error, where D is a random zero-mean Gaussian deviate of unit variance (N(0,1)), and σ_1 is the standard deviation. If σ_1 is chosen as

$$3 \times \sigma_1 \times 100\% = P\% \tag{5.8}$$

then (5.7) represents a Gaussian distributed multiplicative erro with a maximum percent error of P%. Since more than 99% of the Gaussian deviates are within three standard deviations, this maximum percent error definition accurately describes

all but less than 1% of the errors. The model in (5.7) is used to describe all multiplicative spatial error sources in the OLAP, specifically for (5.1) (input spatial gain errors) and for the first term in (5.5) (detector spatial response errors). The δ_i^1's and the δ_i^3's are both determined by $\sigma_1 D$, where σ_1 is separately calculated from (5.8) for each error source. A 10-component random vector from an N(0,1) distribution is used to determine each D value for the 10 point modulators; a 32-component N(0,1) vector is used to determine the D values for the 32 detector elements. The fixed spatial random vector values (deviates) define variations across space. The deviates represent the residual errors after adjustments. They are fixed throughout all OLAP simulation steps T_1 once they are determined.

The P_2 error sources (acoustic attenuation and both types of crosstalk) are modelled as explained previously in (5.2) and (5.4). The α and x_i values in (5.2) are determined by the multi-channel AO cell specifications, and the proposed OLAP construction. Since we propose to use the full 5 μs of time aperture of the AO cell at P_2, the length of each channel is

$$4.2 \text{ mm}/\mu\text{sec} \times 5 \ \mu\text{sec} = 21.0 \text{ mm} \qquad (5.9)$$

where 4.2 mm/μsec is the velocity of sound in the TeO_2 longitudinal mode AO cell. Each channel is thus divided into 10 regions, each of length 2.1 mm. The x_i's are listed in Table 5.2 below, where x_1 is zero as explained earlier.

Different α values in dB/mm will be used in the simulations. They represent various degrees of residual α corrections since acoustic attenuation is deterministic and can be corrected for one frequency operation.

The ratio I_2/I_1, chosen in (5.4) determines the crosstalk level modelled. The value in dB of isolation is calculated from (5.3). The acoustic attenuation and crosstalk error factors are fixed (crosstalk effects are signal dependent, but the level is fixed), and thus their modelling does not require any random inputs.

The remaining errors are the time-varying noise and dark current P_3 errors.

AO Cell Region Center	Position In AO Cell
x_1	0.0 mm
x_2	2.1 mm
x_3	4.2 mm
x_4	6.3 mm
x_5	8.4 mm
x_6	10.5 mm
x_7	12.6 mm
x_8	14.7 mm
x_9	16.8 mm
x_{10}	18.9 mm

Table 5-2: AO Cell Region Center Locations

which are additive errors. These errors are always present in the detector elements, and are not signal dependent (they do not depend on the magnitude of the detector input). They are relative to the detector's dynamic range, or full-scale. These characteristics are evident from our simulation model below, which is similar to that of the multiplicative errors. These errors are considered to have a Gaussian distribution because of their natural random nature. Time-varying detector noise is white and Gaussian, as explained in [38]. A new N(0,1) vector is used to model this at each T_1.

An additive error applied to a quantity z yields $\hat{z}$ where

$$\hat{z} = z + \sigma_2 D \tag{5.10}$$

and a random additive error of $\sigma_2 D$ is used, where D is a N(0,1) deviate and σ_2 is the standard deviation. If the relation

$$3 \times \sigma_2 \times 100\% = P\% \times FS \tag{5.11}$$

is used to determine σ_2, then (5.10) represents a Gaussian distributed additive error, with a maximum percent error of P% of full-scale, where FS is the full-scale range of the device being modelled. From (5.10), it is apparent that this error does not depend on the input value z, but rather it depends on the full-scale dynamic range of the device.

The form in (5.10) is used in our simulator to model the last two terms in (5.5), time-varying noise and dark current. The $n_j(t)$'s and d_j's are all determined by $\sigma_2 D$, where σ_2 is separately calculated from (5.11) for both errors. Since the $n_j(t)$'s vary with time, they will be different for all 32 detector elements in each processor cycle. Thus, every T_1 a new 32-component N(0,1) random vector is generated to determine the D values for the 32 $n_j(t)$'s. A 32-component N(0,1) random vector is generated to determine the D values for the 32 d_j's. Since dark current is fixed, these deviates remain unchanged throughout the OLAP simulation. This simulation modelling scheme thus appropriately distinguishes the time-varying errors from the fixed spatial errors.

We recognize that detector dark current is not properly modelled with a zero-mean distribution. Since dark current effects are additive, and will always contribute positively to the detector element's output, the deviates must be unipolar. The dark current modelling will be changed appropriately in future work. However, the modelling used here is still sufficient and appropriate given the binary nature of the processor. Because of full-scale A/D clipping (to be explained next in this section), a sufficiently large positive dark current deviate could only change a 0 to a 1, and a sufficiently large negative deviate could only change a 1 to a 0 (with the zero-mean modelling used in this report). With the proper modelling (positive deviates only), a sufficiently large positive deviate could change a 0 to a 1, but these deviates should occur twice as often then in the previous case. Thus the net amount of bit errors would be the same.

5.2.6 A/D Error Modelling

One other OLAP error source must still be considered and modelled in our simulator. This is the A/D error for the 32 A/D's that follow each detector, as shown in Figure 4-8. Analog-to-digital conversion of electrical signals is not an exact operation. Specifically, some uncertainty exists in the digital output obtained when the analog signal falls within a certain range between the levels corresponding to two adjacent digital values. For example, if 1 volt represents a digital 0, and 2 volts represents a digital 1, a signal of 1.99 volts will practically always be properly converted to a digital 1. However, a signal of 1.60 volts will not always be converted to a digital 1 (as it should be because it is closer to 2 volts than 1 volt). However, we will assume that an analog level of 1.60 volts is converted to a digital 1 more often than it is converted to a digital 0.

The following A/D error model used was derived to fit the observed behavior of many A/D converters. It is more than adequate for our OLAP simulation. We begin our A/D model by assuming a range of values around each digital level, where conversion to the proper digital number is guaranteed. In the intermediate range, between the guaranteed ranges, a probability distribution function is used to randomly decide to which digital level the conversion is made. The distribution is properly centered such that values are more often converted to the closest digital level.

The guaranteed range was chosen to be 0.25 of the range between digital levels (on both sides of all digital levels). The output for data in the intermediate 0.5 of the range between levels is governed by a probability distribution function. The probability distribution function must be finite for obvious reasons, and was chosen to be triangular for simplicity. It is centered halfway between digital levels, and goes to zero at the 0.25 points. One side has a slope of +4, the other side of -4. A diagram of this modelling appears in Figure 5-3.

The diagram shows digital levels at 1 and 2 volts for discussion purposes.

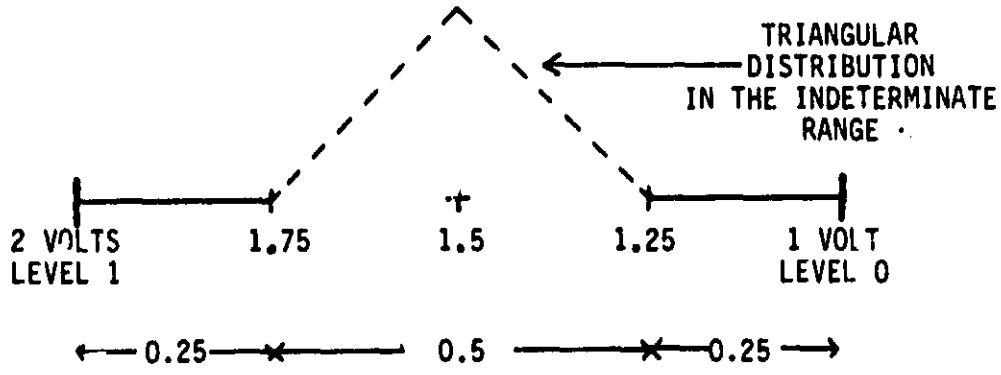


Figure 5-3: A/D Error Modelling

Any signal in the region 1.0 ± 0.25 volts is always converted to a digital 1. The A/D output for inputs between 1.25 and 1.75 volts is controlled by the triangular distribution shown dashed, centered at 1.5 volts. For each A/D conversion (32 every T_1), a random deviate is generated from the triangular distribution. This yields a value between 1.25 and 1.75 volts, and defines the A/D threshold. An input value greater than this threshold is converted to a digital 1, and an input value less than this threshold is converted to a digital 0. The threshold is usually near 1.5 volts, because of the centering of the triangular distribution. This is appropriate for modelling the noise mechanisms in an A/D converter, where an input value will be converted to the nearest digital level most of the time, thus the threshold should be near 1.5 volts most of the time.

In the simulator, the binary bits are represented by real numbers, so that errors may be added to them. The levels for A/D conversion are simply 0 and 1. Also included in the A/D model is the effect of full-scale clipping at the ends of the A/D range, as would be the case with real A/D's. For example, if Figure 5-3 represented the full range of the A/D (a 1-bit A/D), any signals above 2 volts would be converted to a digital 1, and any signals below 1 volt would be converted to a digital 0.

Depending on the type of analog-to-digital converters used, sample-and-hold circuits may be present before the A/D's. This circuitry will introduce only small errors and thus is not considered. Aperture time error will not be of concern since the detector output will not be changing during the sample-hold period T_1. The only other error to be considered is acquisition time error. However, this error is small and can be combined with the time-varying detector noise, since it is additive and changes with time.

We recognize that our A/D model is not as exact as it could be, however it is more than adequate for our simulations. There is never an actual guaranteed range around digital levels, as a finite uncertainty always exists. The A/D error is better modelled by a Gaussian, with some type of tail truncation to make the distribution finite. It would cover the entire range between digital levels. Our triangular distribution modelling is simpler, and a Gaussian model would not create a significant difference in simulation results. However, the A/D error model will be changed appropriately in future work.

5.3 Olap Simulation Software Details

The software for simulating the OLAP and its optical errors was written in DEC Fortran for the VAX 11/750. It consists of four programs, a main program and three subroutines, plus various IMSL [39] noise generation routines. The programs simulate the OLAP operation and model its errors. The LU decompositon algorithm for the OLAP, as described in Chapter 4, is implemented. The binary encoded data in all M=10 processor channels and N=32 AO cell channels is explicitly represented within the programs. The four programs are named FINSIM, DCTOBN, OPTPRC, and ECLSFT.

The main program is FINSIM, which reads the finite element case study data. It then reads in the OLAP processing and error modelling inputs. The OLAP processing inputs consist of two parameters: the number of bits to be used (this is variable for future use), and the power-of-two weighting of the most significant

bit. The number of bits is 32 for all simulations. The weighting of the MSB will be 2^{10} for all case study simulations, and thus the weighting of the LSB was 2^{-21}. This choice accounts for numbers between 2^{-21} and $\simeq 2^{11} \simeq 2047.9999$ and is chosen because it matches the range of data in our finite element problem.

There are eight variable error model inputs, one for each error source, and a starting seed value for the IMSL random number generation routines. Only one seed is needed because a new seed is returned from each IMSL subroutine call, which is used for the next IMSL subroutine call. For fixed errors, the same N(0,1) deviates are used at each T_1. For time-varying detector noise, a new N(0,1) deviate is used each T_1. The first error modelling input specifies the P_1 input spatial gain maximum percent error (MPE). The second input specifies the P_3 detector spatial response maximum percent error (MPE). The third and fourth inputs specify the levels of detector time-varying noise and dark current respectively, as maximum percent of full-scale errors (MPFSE). The fifth input specifies the acoustic attenuation of the multi-channel AO cell in dB/mm. The sixth and seventh inputs specify the crosstalk isolation (positive dB values) in dB for adjacent and even-odd channel crosstalk, respectively.

The final error modelling input is the starting seed for the IMSL noise generation routines. Only one seed is required, as each IMSL subroutine generates a new seed for use on the next subroutine call. This seed is kept the same for all simulations. The seed determines the deviates for fixed and time-varying noise, which are generated differently as noted above. It is necessary that the same deviates are generated each simulation, because the deviate values create the errors within the processor. Thus, for meaningful and significant comparisons of errors and error levels in different simulations, the starting seed is kept the same.

The subroutine DCTOBN is used to convert all of the real-valued elements in the finite element case study matrix equation to 32-bit fixed-point binary

representations (with the MSB representing 2^{10}). The OLAP processes 32-bit magnitude data and handles bipolar data using sign-magnitude representation as explained in Chapter 4. Therefore, DCTOBN actually creates 33-bit sign-magnitude representations, with the extra bit being a sign bit. An exclusive-or of the sign bits of the non-zero P_1 input, and the number in the corresponding region of the AO cell at P_2, is formed to determine the sign of the product (in OPTPRC).

The OLAP operates with the bits of new numbers fed to the inputs every T_2. The bits of each number in the multi-channel AO cell are fed in parallel and move into a new region of the cell every T_2. Each T_2 consists of 32 bit times, T_1. Every T_1, a new bit in each number is presented to the P_1 inputs, while the data in the AO cell moves a small amount but remains in the same T_2 region. Every T_1, the detector values are A/D'ed and shifted, to perform the convolution.

The program FINSIM implements the LU decomposition algorithm. It forms the decomposition matrices, and performs the necessary matrix-matrix multiplications. Each n by n matrix-matrix multiplication is performed as n^2 vector inner products, with one VIP performed on the OLAP every T_2, as detailed in Chapter 4. Each T_2, FINSIM calls the subroutine OPTPRC, which receives the 10 binary numbers fed to the P_1 inputs, and the 10 binary numbers present in the P_2 AO cell. OPTPRC implements the 32 T_1 cycles (the 32 convolutions and partial product sums) and applies all seven optical error sources to the binary data. The diagonal partitioning is explicitly performed. The main diagonal is partitioned separately and processed, then the remaining 14 lower diagonals (since P^k is lower triangular) are partitioned in 10-diagonal increments, requiring 3 partitions per matrix.

The IMSL routine GGNML is used to produce 74 Gaussian deviates: 10 for the P_1 input spatial gains, 32 for the P_3 detector spatial responses, and 32 for the P_3 detector dark currents. Since these are fixed for the OLAP operation, they are generated once in FINSIM and passed to OPTPRC each T_2. GGNML is also used to generate the 32 time-varying detector noise Gaussian deviates each

T_1 within OPTPRC. OPTPRC thus calls the subroutine ECLSFT 32 times each T_2, once at the end of every T_1. ECLSFT simulates the A/D process and the shift-add process in the P_3 detector array. The A/D process is simulated as described in section 5.2. A triangular distribution deviate is generated for each of the 32 A/D conversions each T_1 by the IMSL routine GGTRA.

OPTPRC converts the mixed binary vector inner product to a real number and returns it to FINSIM. This represents one element of a matrix-matrix multiplication. A real number is returned for storage and handling convenience, rather than a 33-bit binary representation. This is a result of multiplying each mixed binary bit (of the 63 from the 32-bit convolution) by its appropriate power of two and adding them. DCTOBN is used to convert the real value to the 33-bit sign-magnitude binary representation when it is subsequently needed. In the actual OLAP, the digital system would always use the binary values.

The final result from the LU decomposition algorithm is the upper-triangularized system of equations, (4.20). The OLAP is intended to produce (4.20), since the triangularization process represents most of the computations required to solve (4.19). Equation 4.20 would then be solved in digital hardware, since it is such a trivial operation. It should be pointed out that triangular systems of equations can also be solved with the standard optical processor of Figure 4-1, as described in [26]. An adaptation of that algorithm should be possible for our binary encoded OLAP. Once FINSIM completes the LU decomposition algorithm and produces the triangular matrix equation (4.20), it is then solved digitally for the unknown DOFs.

Some remarks about the programming and running of the simulator are now advanced. First, the coding of the LU decomposition algorithm with partitioning is quite complicated. The indexing for the data flow, as diagrammed in Figure 4-9 requires very careful bookkeeping. The simulation program also requires a large amount of computer time. Solving a 24 by 24 case study matrix equation using our OLAP simulator averages 4 hours of CPU time on a VAX 11/750, with only single precision.

To detail the steps required, we consider a 24 by 24 case study (problem of chapter 3 with the boundary conditions noted in subsection 4.5.3), and we note that the LU decomposition algorithm requires

$$[24^2+23^2+\ldots+3^2+2^2] + [24+23+\ldots+3+2] = 4899+299 = 5198 \qquad (5.12)$$

vector inner products: 4899 VIPs for the matrix-matrix multiplication in (4.13), and 299 VIPs for the matrix-vector multiplication in (4.13), where the multiplications in both equations are reduced in dimensionality by one every step (there are N-1 steps). With P^k being partitioned 3 times as explained earlier in this section, this number must be multiplied by 3 to give $3\times5198=15594$ total VIPs for the LU decomposition of the case study. This is the number of T_2's simulated, and the number of times OPTPRC is called from FINSIM. OPTPRC must perform each of the scalar-vector (1-bit scalar in P_1, 32-bit vector in P_2) multiplications serially to properly simulate the optical errors. Each VIP (every T_2) thus requires 10 real multiplications (since M=10) of the 32-bit vectors, or 320 multiplications every T_1. Since there are 32 T_1s in every T_2, each VIP requires 320×32 multiplications. Thus, a total of $15594\times320\times32 = 160\times10^6$ multiplications must be performed serially in the simulator for the LU decomposition of the case study. In addition, every T_1 the shift, adds, noise generations, noise additions, and A/D conversions (requiring a call to GGTRA) must be performed, not to mention all the program overhead and control. Thus, to complete approximately 100 simulation runs necessary to fully evaluate the OLAP performance with optical errors, over 2 weeks of CPU time will be required.

5.4 Initial Simulation Results

This section describes and discusses the results of simulating the multiplication of three pairs of 32-bit numbers. We expect the OLAP to be able to tolerate significant optical errors because of the binary encoding used. This occurs because the processor need only represent two levels. The errors that limit analog processors to 30-40 dB of dynamic range can thus be substantially larger before they will adversely affect the binary data in our OLAP, which only requires 3 dB of dynamic range.

The effect of the optical errors is significantly dependent on the position of the 1's and 0's within each number in the processor, and how many 1's and 0's there are. This occurs because the multiplicative errors and acoustic attenuation have no effect on 0's. If the power-of-two weighting of the most significant bit (10 for our simulations) is varied, the simulation results with optical errors will vary because the locations of the 1's and 0's will change in the fixed point representation. The effects of both types of crosstalk will also be influenced by the positions of the 1's and 0's.

The dependency of processor errors on the position and number of 1's and 0's is now demonstrated, and it is quantified by the data in Tables 5.3, 5.4, and 5.5. Each table contains entries which show the results of a single 32-bit multiplication on the OLAP. Each multiplication requires one T_2, i.e., 32 T_1's. The simulations were performed with the previously described software, slightly modified so that one 32-bit number is input to point modulator 1, and the other input data is in region 1 of the P_2 AO cell channels. The same starting seed was used for all runs.

For each test, an individual error source is added, and the two numbers are multiplied with the OLAP simulator. The result of a 32-bit multiplication by digital convolution of two numbers is a 63 bit mixed binary number. In each test, the number of mixed binary bit errors resulting from the OLAP multiplication is tabulated. The 63 mixed binary bits (as explained in section 4.3) are the sum of 32 shifted sets of c_j outputs (1 set every T_1) after A/D conversion to binary values. The number of bits that differ from the 63 mixed binary bits produced by an error-free OLAP multiplication are tabulated The size of the differences is not considered (most differed by one or two bits). The individual tests are referred to by their test number on the left-hand side of each table.

The number and position of the 1's and 0's in the 32-bit binary numbers depends on 1) the number itself, and 2) the power-of-two weighting given to the

most significant bit. The latter is included under the heading "MSBASE" in the tables. Another factor affecting the positions of the 1's and 0's in the OLAP is 3) which number is the multiplicand fed to the multi-channel AO cell at P_2, and which number is the multiplier fed into the point modulator in P_1. Thus, to illustrate the affect of changing positions and different numbers of 1's and 0's in the OLAP, the three factors just mentioned were varied to produce the results in Tables 5.3, 5.4, and 5.5.

The data in each table corresponds to the multiplication of one of three different pairs of 32-bit numbers, each shown in Figure 5-4. The three pairs of numbers exhibit different features. The first pair of numbers (Table 5.3 data) have about an equal and randomly distributed number of 1's and 0's. The numbers used in the Table 5.4 tests represent an extreme case where many 1's are present and close together in both numbers. The numbers for Table 5.5 represent another extreme case where there are very few 1's in each number separated by many 0's. The variations between the three pairs of numbers account for variations in factor 1 above.

The second and third factors above are varied within each table. For each individual error source, tests include the error source levels that yielded no bit errors, and other error source levels up to those that yielded many bit errors. This is done for each error source separately for MSBASE=10. Variations in the multiplicand and multiplier were also considered in each table. In each table, the first number (fed to the P_2 AO cell) is the multiplicand and the second number (fed to the P_1 point modulators) is the multiplier. Other variations in the last two factors (MSBASE and the order of the multiplicand/multiplier) are included in each table.

Examination of the entries in Tables 5.3 - 5.5 readily shows how the position of 1's and 0's in the OLAP affects the bit errors. Note that some of the MPE's and MPFSE's assumed are extraordinarily large in order to introduce errors. This

Table 5.3:

237.104 - 0 0 0 1 1 1 0 1 1 0 1 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0
78.655 - 0 0 0 0 1 0 0 1 1 1 0 1 0 1 0 0 1 1 1 1 0 1 0 1 1 1 0 0 0 0 0 0

(a)

Table 5.4:

1023.9921875 - 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1787.953125 - 0 0 0 0 0 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0

(b)

Table 5.5:

528.0078125 - 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
256.015625 - 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

(c)

Figure 5-4: Multiplication Pairs for Tables 5.3 - 5.5

is because we are dealing with one multiplication as defined, thus, only a single fixed input spatial gain deviate, and 32 fixed detector spatial response deviates are used. Simulation results for multiplication of many numbers, as in the case study solution simulations that will be performed in the near future, are expected to yield reasonable and predictable levels of tolerable optical errors, which will be significantly larger than the error levels present in analog optical processors. The intent here is only to show the effect of the positions of 1's and 0's in the OLAP.

Each table contains a test number in the left-hand column to refer to the test described in that row of the table. The first column contains the multiplicand, the second contains the multiplier, and the third contains the value of MSBASE used for that test. In Tables 5.4 and 5.5, only the digits to the left of the decimal point are tabulated in the multiplicand and multiplier columns. In each test

only one individual error source is used. Thus, only one entry will appear in one of the next five columns, which shows the MPE, MPFSE, or isolation level of its corresponding error source. Acoustic attenuation was not included because demonstration of its effects requires more than one multiplication (i.e., more than one P_2 AO cell region must be used). Only adjacent channel crosstalk was included. The final column contains the number of mixed binary bit errors resulting from the multiplication.

The effect of factor 1) on the number of bit errors is easily seen by comparing similar tests of different tables, such as tests 1.3, 2.3, and 3.1. In test 1.3 a 60% input spatial gain error resulted in 15 mixed binary bit errors. In test 2.3, a 60% input spatial gain error with a different pair of numbers being multiplied yielded 12 mixed binary bit errors in the product. However, in test 3.1 the multiplication with the same input spatial gain error yielded 0 bit errors. This comparison clearly shows how multiplicative error effects differ when the number of 1's and 0's differ. Since input spatial gain is a multiplicative error, it does not affect 0's, but only affects 1's. The numbers in tests 1.3 and 2.3 both have a substantial amount of 1's, and thus both tests yielded many bit errors. In test 3.1, the numbers being multiplied have very few 1's, thus the multiplication was not affected by the multiplicative error.

The effect of factor 2) on the number of bit errors can be observed by comparing corresponding tests within the tables, such as 1.16 and 1.17, or 2.26 and 2.30. Test 1.16 involves MSBASE=10 and a 50% detector time-varying detector noise error. It resulted in 7 mixed binary bit errors. In test 1.17, MSBASE was change to 14, which moved the positions of the 1's and 0's in the 32-bit fixed-point representations. This resulted in the 50% time-varying detector noise having a smaller impact on the multiplication, producing only 5 mixed binary bit errors. This is because the 32 c_j values at each T_1 appear on different detectors (a difference of 4 physical detectors since 14-10=4), and thus the detector error has a different effect on the resultant convolution. Similarly, tests

Table 5-3: Individual Multiplication Error Simulation Results - Average Case

TEST NUMBER	AO CELL P_2 NUMBER	INPUT ARRAY P_1 NUMBER	MSBASE	INPUT SPATIAL GAIN MPE	DETECTOR SPATIAL RESPONSE MPE	DETECTOR TIME-VARYING NOISE MPFSE	DETECTOR DARK CURRENT MPFSE	ADJACENT CH. CROSSTALK ISOLATION dB	MIXED BINARY BIT ERRORS
1.1	237.104	78,655	10	50					0
1.2	"	"	"	55					5
1.3	"	"	"	60					15
1.4	"	"	"	70					23
1.5	"	"	"	80					30
1.6	78,655	237,104	"	55					5
1.7	"	"	"	60					10
1.8	"	"	10		75				0
1.9	"	"	"		80				1
1.10	"	"	"		85				1
1.11	"	"	"		105				3
1.12	"	"	"		125				6
1.13	"	"	14		105				9
1.14	"	"	10			30			0
1.15	"	"	"			40			1
1.16	"	"	"			50			7
1.17	"	"	14			50			5
1.18	78,655	237,104	10			50			6
1.19	237,104	78,655	10				35		0
1.20	"	"	"				40		1
1.21	"	"	"				50		2
1.22	"	"	"				60		10
1.23	"	"	7				60		18
1.24	78,655	237,104	10				60		22
1.25	237,104	78,655	10					5	33

Table 5-4: Individual Multiplication Error Simulation Results - Adjacent 1's

TEST NUMBER	AO CELL P_2 NUMBER	INPUT ARRAY P_1 NUMBER	MSBASE	INPUT SPATIAL GAIN MPE	DETECTOR SPATIAL RESPONSE MPE	DETECTOR TIME-VARYING NOISE MPFSE	DETECTOR DARK CURRENT MPFSE	ADJACENT CH. CROSSTALK ISOLATION dB	MIXED BINARY BIT ERRORS
2.1	1023	1787	10	40					0
2.2	"	"	"	50					2
2.3	"	"	"	60					12
2.4	"	"	16	60					12
2.5	1787	1023	10	60					11
2.6	1023	1787	"	"	70				0
2.7	"	"	"	"	80				1
2.8	"	"	"	"	90				3
2.9	"	"	"	"	100				4
2.10	"	"	18	"	90				8
2.11	"	"	10	"	"	30			0
2.12	"	"	"	"	"	40			1
2.13	"	"	"	"	"	50			7
2.14	"	"	"	"	"	60			13
2.15	"	"	20	"	"	30			0
2.16	"	"	"	"	"	40			2
2.17	"	"	"	"	"	50			8
2.18	"	"	"	"	"	60			15
2.19	1787	1023	10	"	"	30			0
2.20	"	"	"	"	"	40			0
2.21	"	"	"	"	"	50			6
2.22	"	"	"	"	"	60			13
2.23	1023	1787	"	"	"	"	30		0
2.24	"	"	"	"	"	"	40		1
2.25	"	"	"	"	"	"	50		4
2.26	"	"	"	"	"	"	60		12
2.27	"	"	20	"	"	"	30		0
2.28	"	"	"	"	"	"	40		0
2.29	"	"	"	"	"	"	50		4
2.30	"	"	"	"	"	"	60		17
2.31	1787	1023	10	"	"	"	30		0
2.32	"	"	"	"	"	"	40		1
2.33	"	"	"	"	"	"	50		3
2.34	"	"	"	"	"	"	60		10
2.35	1023	1787	10	"	"	"	"	5	1

Table 5-5: Individual Multiplication Error Simulation Results - Few 1's

TEST NUMBER	AO CELL P_2 NUMBER	INPUT ARRAY P_1 NUMBER	MSBASE	INPUT SPATIAL GAIN MPE	DETECTOR SPATIAL RESPONSE MPE	DETECTOR TIME-VARYING NOISE MPFSE	DETECTOR DARK CURRENT MPFSE	ADJACENT CH. CROSSTALK ISOLATION dB	MIXED BINARY BIT ERRORS
3.1	528	256	10	60					0
3.2	"	"	"	70					1
3.3	"	"	"	80					2
3.4	"	"	"	90					4
3.5	"	"	"	100					5
3.6	"	"	20	60					0
3.7	"	"	"	70					2
3.8	"	"	"	80					2
3.9	"	"	"	90					3
3.10	"	"	"	100					4
3.11	256	528	10	70					0
3.12	"	"	"	80					1
3.13	"	"	"	90					2
3.14	"	"	"	100					4
3.15	528	256	10	70	400				0
3.16	"	"	"	"	500				2
3.17	"	"	"	"	2000				2
3.18	"	"	20	"	200				0
3.19	"	"	"	"	400				2
3.20	"	"	"	"	900				4
3.21	"	"	"	"	2000				4
3.22	"	"	10	"		30			0
3.23	"	"	"	"		40			1
3.24	"	"	"	"		50			7
3.25	"	"	"	"		60			14
3.26	"	"	20	"		30			0
3.27	"	"	"	"		40			1
3.28	"	"	"	"		50			7
3.29	"	"	"	"		60			13

CONTINUED:

TEST NUMBER	AO CELL P_2 NUMBER	INPUT ARRAY P_1 NUMBER	MSBASE	INPUT SPATIAL GAIN MPE	DETECTOR SPATIAL RESPONSE MPE	DETECTOR TIME-VARYING NOISE MPFSE	DETECTOR DARK CURRENT MPFSE	ADJACENT CH. CROSSTALK ISOLATION dB	MIXED BINARY BIT ERRORS
3.30	256	528	10				30		0
3.31	"	"	"				40		1
3.32	"	"	"				50		7
3.33	"	"	"				60		14
3.34	528	256	10				30		0
3.35	"	"	"				40		1
3.36	"	"	"				50		6
3.37	"	"	"				60		22
3.38	"	"	20				30		0
3.39	"	"	"				40		1
3.40	"	"	"				50		6
3.41	"	"	"				60		22
3.42	256	528	10				30		0
3.43	"	"	"				40		1
3.44	"	"	"				50		6
3.45	"	"	"				60		22
3.46	528	256	10					5	0

2.26 and 2.30 show the same type of change, except that 60% detector dark current is the error source that is used. Tests 2.8 and 2.10 exhibit the same bit error difference with 90% spatial detector response error.

The effect of factor 3) can be observed by comparing tests such as 1.22 and 1.24, or 2.3 and 2.5, or 3.4 and 3.13. In tests 1.22 and 1.24 a 60% detector dark current error is used, and MSBASE=10 for both. The tests differ in that the multiplicand and multiplier are switched between the two. This means that the 1's and 0's involved in the multiplication are represented in different planes of the OLAP between the two tests. Thus, a different number of bit errors is expected because the positions of the 1's and 0's is varied greatly. Test 1.22 yielded 10 mixed binary bit errors, while test 1.24 yielded 22 bit errors. Similar results are observable between the other tests mentioned above, and elsewhere in the tables.

Some other interesting, yet predictable, results can be observed from these tables. In Table 5.5, the two numbers have few 1's. Thus, a certain amount of a given additive P_3 error should cause about the same number of bit errors, for given variations of factors 2) and 3). This occurs because there are so few 1's that the additive noise almost always has the same effect on the 0's, since the 32 output c_j values every T_1 will be mostly zeros. This behavior can be seen in tests 3.22-3.33 for detector time-varying noise, and in tests 3.34-3.45 for detector dark current. For example, tests 3.24, 3.28, and 3.32 are all for 50% MPFSE detector time-varying noise. Each test is different from the others in either the multiplicand/multiplier or MSBASE (10 or 20), however, they all yield 7 mixed binary bit errors. It is not explicitly indicated in Table 5.5, but the bit errors occurred in the same 7 bits in each test.

Tests 3.15-3.21 show another interesting phenomenon. The detector spatial response error is multiplicative and thus only affects the very few 1's that are in the numbers being multiplied, and hence produced on the detector elements. Even

then, the only errors produced are those due to negative deviates. This is due to the full-scale A/D clipping, where any positive deviate which increases a 1 will have no net effect because the resultant value will be clipped to a full-scale value of 1 during the A/D conversion every T_1. In each test, the deviates were small and very large spatial detector MPE levels were needed to produce bit errors. The errors were in the same bits (as expected because there are so few non-zero bits that can be affected), caused by changing a 1 to a 0.

Another interesting set of results can be seen by comparing the adjacent channel crosstalk tests in each table: 1.25, 2.35, 3.46. When 5 dB crosstalk was imposed on the Table 5.5 numbers (test 3.46), no mixed binary errors resulted. This is because there are so few 1's in the multiplicand which is in the P_2 AO cell. The first intuitive notion is that the same amount of crosstalk applied to the Table 5.3 and 5.4 numbers would surely result in more bit errors for the Table 5.4 numbers because there are a lot of 1's adjacent to each other, which should be detrimental in terms of adjacent channel crosstalk. However, many more bit errors are actually produced for the numbers of Table 5.3 (test 1.25). A valid explanation emerges after thinking about the crosstalk error producing mechanism. Since crosstalk is additive and positive, because only 0's and 1's are represented, it only affects 0's because anything added (positively) to a 1 gets clipped to a 1 by the A/D conversion full-scale clipping, as explained in section 5.2. The 0's that are affected by adjacent channel crosstalk are those adjacent to 1's. It is clear that there are many more such 0's in the numbers of Table 5.3, than there are in the numbers of Table 5.4. These remarks, however, are only valid when one number is in the AO cell at P_2. When 10 numbers are present in the AO cell (as there would be for normal operation), the crosstalk effect on one bit is dependent on the 10 data bits in each of the contributing AO cell channels. Only zeros are still affected, but other numbers contribute to the crosstalk as well.

5.5 Summary and Conclusion

This chapter has presented the results of initial error source simulations on our optical linear algebra processor. A 10-channel OLAP was simulator was developed because it represents a general purpose banded matrix-vector processor. The error modelling used in the simulation program was presented and each error source was described. The A/D conversion modelling was detailed, and the simulation software was described.

Optical error simulations were performed for the multiplication of three pairs of 32-bit numbers. The results of the simulations of each multiplication were given in three separate tables. Comparisons of the results were discussed, and they revealed the error producing mechanisms that operate in a digitally encoded optical processor. It was shown how any errors produced are dependent on the numbers being multiplied, which determines the number of 1's and 0's in the processor, and the selection of MSBASE and the choice of the multiplier/multiplicand, which determines where the 1's and 0's are positioned in the processor.

6. Summary and Future Work

6.1 Summary

This report has presented a new digitally encoded optical processor. The research focused on demonstrating the architecture's usefullness for finite element problems, or banded matrix-vector problems that require high accuracy processing. The proposed optical linear algebra processor computes with 32-bit accuracy and performs multiplications by digital convolution. The processor performs a 10-element vector inner product every T_2, which is 0.5 μsecs. From section 5.3, the LU decomposition algorithm implemented required 15594 T_2's to compute a triangularized system of equations of size N=24. Thus, the OLAP would require 15594×0.5 μs = 7.80 ms to compute the triangularized equations. The processing is performed (with 32-bit accuracy and existing components) at a rate of 2.0×10^7 multiplications and 1.8×10^7 additions per second.

The finite element method was described in Chapter 2 for structural mechanics problems. A finite element equation derivation example was given for a triangular plane strain element. The matrix equation (stiffness matrix) properties were detailed and solution methods were discussed. Remarks on solving nonlinear and dynamic problems were included at the end of the chapter. The case study was detailed in Chapter 3. It described the discretization of an aluminum plate with eight plate bending finite elements. The stiffness matrix assembly process was detailed and its properties were quantified. Chapter 4 examined the weaknesses of analog optical processors and presented our digitally encoded processor architecture. The processor operation was detailed for existing components, including a 10-channel AO cell (input P_1 array) and a 32-channel AO cell (P_2). Implementation of banded matrix-vector products was described, and a direct LU decomposition algorithm using only banded matrix-vector products was developed for implementation of our processor. The use of a 1-channel system for the LU decomposition algorithm was detailed, and the subjects of large finite

element and large matrix problems in general (using matrix partitioning) were addressed. Finally, Chapter 5 presented the first error simulation program for a digitally encoded optical processor. The error sources and error modelling were presented, and the software was described. Initial simulation results were presented and discussed.

6.2 Future Work

This report described an optical error simulation program for solving a set of finite element equations on the OLAP. Simulation results were only described for individual multiplications. This was due to the large amount of CPU time required for the simulation runs when solving a case study. The next phase of this research is to fully evaluate the OLAP's performance with optical errors, and as noted earlier, this will require approximately 100 simulation runs. The goal of the evaluation will be to determine what levels of the various optical errors can be present in the processor, and still yield accurate solution results. It is expected that the processor can tolerate a large amount of optical errors, given its reduced dynamic range requirements. Some indication of the tolerable error levels is available from the initial simulation results in this report, however many of those error levels are expected to be higher than tolerable levels for a complete problem solution, since only single multiplications were performed.

There are many related topics to be investigated. One of these topics, mentioned in Chapter 4, is handling bipolar data with a twos complement representation. There are many possibilities for such an representation on our OLAP and similar architectures. The next step in number representation is to implement floating point processing. Up to now, all optical processing has been described for fixed point operations. Floating point representations will require some sophisticated architectures and/or algorithms and implementations.

Another general topic for future work is implementation of other algorithms on the OLAP, and considering similar or different processor architectures. Of special

interest are iterative (indirect) algorithms, which will be discussed in more detail below. Many other direct and indirect algorithms can be developed for OLAP implementation. Presently, our OLAP was used to solve finite element problems formulated with a banded matrix. Algorithms using other kinds of matrix formats may prove useful, such as profile storage described in section 2.7. Specialized algorithms for use on dedicated special-purpose finite element machines may be very efficient for optical processors. Some of these algorithms are described in [34]. This report focused on the solution of a linear static finite element problem only. The solution of nonlinear and dynamic problems (section 2.8) is also a large future work topic. Other solution techniques could be implemented for solving these types of problems.

The OLAP described in this report does not use frequency multiplexing in the multi-channel AO cell at P_2. Frequency multiplexing is another dimension that can be made available on the processor by using F frequencies in P_2, and placing F linear detector arrays in P_3 of Figure 5-1. This is certainly a topic for future work. The F linear detector arrays would be stacked in the x direction of Figure 5-1. If F frequencies are used, then F vector inner products are obtained in P_3, which represents a matrix-vector product. All linear algebra operations can be performed in terms of the basic vector inner product and matrix-vector product operations. Thus, although we do not propose using frequency multiplexing for solving finite element problems, any of the algorithms employing frequency multiplexing for the processor of Figure 4-1 are applicable to our digitally encoded OLAP of Figure 4-6.

Two future work topics are very pertinent to the issues this report has discussed. They are iterative (rather than direct) algorithm implementations and encoding in radices greater than two (binary). They have both been mentioned throughout the report, but will briefly be discussed in more depth below.

6.2.1 Iterative Algorithms

Two solution methods are possible for solving systems of linear algebraic equations: direct and iterative (indirect). Section 2.7 discussed how iterative algorithms are also useful for solving finite element systems of equations. A direct solution technique, the LU decomposition method, was chosen as the equation solver in this report because of the widespread use of similar algorithms in many finite element programs for digital computers. It is important to consider the implementation of iterative algorithms on our OLAP.

There are two major drawbacks to solving finite element problems with iterative methods. The first is that they require an indeterminate amount of computations (iterations) to converge to an acceptable solution, whereas with a direct method, a fixed number of computations is required. Second, the entire iterative solution process must be repeated for a different right-hand side (i.e. the **p** load vector) when the matrix **K** remains the same. With a direct method, the triangularization of **K** need only be performed once for multiple **p** vectors. However, iterative methods have three significant advantages over direct methods: they are easier to program, they require less storage, and they are more tolerant of errors.

The previous remarks are fairly standard for comparing direct and iterative solvers. Some other remarks should be made in light of our concerns. Equation 2.21 defines a standard Gauss-Seidel iterative algorithm. A more basic general iterative procedure can be defined for the solution of

$$\mathbf{Ab} = \mathbf{c} \tag{6.1}$$

as follows

$$\mathbf{b}_{i+1} = \omega(\mathbf{c}-\mathbf{Ab}_i) + \mathbf{b}_i \tag{6.2}$$

where $\mathbf{b}_i$ is the solution vector at iteration i. An initial estimate $\mathbf{b}_0$ is used to start the algorithm. The acceleration parameter ω may vary with i and/or depend on the previous $\mathbf{b}_i$ values. Other $\mathbf{Ab}_i$ products at other steps i may also be included in the parentheses.

The important point is that (6.2) shows how feedback is used in an iterative solver. Past $\mathbf{b}_i$ vectors are used to form the new $\mathbf{b}_{i+1}$ vectors. This $\mathbf{b}_{i+1}$ vector is then fed back through a system to form a $\mathbf{b}_{i+2}$ vector. Since feedback is used, iterative solvers are corrective. If errors are introduced in one $\mathbf{b}_i$ vector, calculation of $\mathbf{b}_{i+1}$ produces a vector that corrects the error towards the true solution. More iterations are required, but the system will eventually recover from the errors (assuming that a relatively stable indirect algorithm is used).

The corrective nature of iterative algorithms is attractive for optical processing as well as digital processing. Since optical errors exist, a more reliable optical processor may be one utilizing an iterative solver, since there is no error recovery built into direct solvers.

There is no way to avoid the second drawback of iterative solvers mentioned above, i.e., the need to repeat the solution process for new multiple right-hand side vectors. However, the first drawback, the indeterminate amount of operations, can be made less severe. Some iterative algorithms have finite bounds on the number of iterations required to achieve a certain solution accuracy, and they are usually defined for ideal processing, i.e., no processing errors. The bounds are dependent on the problem parameters, and most often dependent on a multiplicative fraction of the condition number of the matrix in the matrix equation to be solved. One such bound for an iterative algorithm is described in [37] and [40], where it is proven that solution accuracy within 1% can be achieved with no more than 3C iterations, where C is the condition number of the matrix. The condition number is defined as the quotient of the largest and smallest eigenvalues of the matrix. This is only one example, and many other iterative algorithms and iteration bounds exist. Higher accuracy requires more iterations, within the accuracy limit of the processor.

As an example, the condition number for our case study, with N=24 after boundary conditions as noted in subsection 4.5.3, is quite large. The largest

eigenvalue is 12.212631, and the smallest is 0.000291. Thus, C=41967.8 . With the algorithm mentioned above, approximately 126 thousand iterations would be required for solution accuracy within 1%. This would require over 300 thousand T_2's on our 10-channel OLAP, as compared with the 15595 T_2s required for our direct LU decomposition solution. In general, the number of iterations does not increase appreciably when optical errors are present [35], [40]. Indirect algorithms should be simulated to determine those effects.

Another fortunate aspect of iterative algorithms can be used to appreciably reduce the large number of required iterations. If a good initial estimate of the solution vector can be made, the number of iterations required for acceptable accuracy can be greatly reduced. Of course, a significant reduction in iterations requires a good initial estimate. Certainly, for linear static finite element problems, very good initial estimates can be made. The ability to obtain good initial estimates for nonlinear and dynamic problems depends on the problem and the user's familiarity with similar results. Since most iterative algorithms are quadratically convergent, good initial estimates may create a large decrease in the number of required iterations.

Most of the computational burden of iterative algorithms such as (6.2) are in the matrix-vector multiplication $\mathbf{A}\mathbf{b}_i$. For finite element problems, **A** is banded, and thus only banded matrix-vector products need to be performed. Chapter 4 explained how our OLAP is well suited for those operations. Other types of iterative solvers exist, however many are less attractive because they cannot be written and implemented as simply as (6.2). Some examples are the many conjugate gradient solution techniques. Implementations of these and other indirect solvers have not yet been developed for optical processors. They also surely deserve attention in future work.

6.2.2 Higher Radices

Another topic deserving future attention is encoding data in radices greater than two (binary encoding). The system dynamic range requirements of such schemes were described in section 4.4. We have seen how our OLAP is relatively impervious to optical errors with binary encoding. It seems reasonable that higher radix encoding can be used to allow fewer P_2 channels to be used and with modest output dynamic range requirements, without significantly affecting processing accuracy. Obviously, if a high enough radix were used, the optical errors will affect the solution accuracy as if an analog processor were used.

The big advantage of using a radix larger than two, say radix R, is that less R-bits (as opposed to binary bits) are needed to obtain 32-bit accuracy. Thus, less AO cell channels are required at P_2, and each time period T_2 required for a 10-element VIP would be significantly less than $32T_1$, since 32 bits would not be used in the number representations. This would increase processing speed and reduce the processor size. Those advantages can then be traded off against less optical error tolerance or poorer processing accuracy.

If radix R is used, the following equation can be written

$$2^{32} = R^X \tag{6.3}$$

where X is the number of R-bits required to obtain 32-bit accuracy. Thus if a radix of R=4 was used for data encoding, only X=16 R-bits would be required. This would decrease the P_2 AO cell requirements by a factor of 2, and cut the processing time by half (since T_2 would equal $16T_1$). Other processing requirements need to be detailed, such as the detector array processing. Briefly, if M is the number of input point modulators defined in Chapter 4, and if R is the radix used then

$$M(R-1)^2+1 = L \tag{6.4}$$

where L is the number of digital levels that need to be A/D converted at each detector array element in P_3. Thus, to find the number of bits required for the

A/D's, simply take LOG_2 of the expression in (6.4) and round the result to the next largest integer. These are just initial remarks on the subject of encoding in higher radices, and more work in the subject area will be forthcoming.

7. References

1. R. D. Cook, *Concepts and Applications of Finite Element Analysis*, John Wiley and Sons, 1981.

2. R. H. Gallagher, *Finite Element Analysis, Fundamentals*, Prentice-Hall Inc., 1975.

3. O. C. Zienkiewicz, *The Finite Element Method*, McGraw-Hill, 1977.

4. J. Bielak, Lecture Material, Carnegie-Mellon University, Department of Civil Engineering, 1984.

5. C. A. Brebbia, J. J. Connor, *Fundamentals of Finite Element Techniques*, John Wiley and Sons, 1974.

6. C. A. Fellipa, "Solution of Linear Equations, with Skyline Stored Symmetrix Matix", *Comp. Struct.*, Vol. 5, pp. 13-30, 1975.

7. B. M. Irons, "A Frontal Solution Program", *Int. J. Num. Meth. Eng.*, Vol. 2, pp. 5-32, 1970.

8. D. G. Row, "A Substructure Technique for Nonlinear Static and Dynamic Analysis". Ph.D. Thesis, University of California, Berkely, 1978.

9. N. M. Newmark, "A Method for Computation of Structural Dynamics", *Proc. ASCE*, Vol. 85, EM3, pp. 67-94, 1959.

10. G. Dahlquist, *Numerical Methods*, Prentice-Hall Inc., 1974.

11. K. J. Bathe, E. L. Wilson, "Stability and Accuracy Analysis of Direct Integration Methods", *Int. J. Earthq. Eng. Str. Dyn.*, Vol. 1, pp. 283-291, 1973.

12. J. C. Houbolt, "A Recurrence Matrix Solution for the Dynamic Response of Elastic Aircraft", *J. Aero. Sci.*, Vol. 17, pp. 540-550, 1950.

13. H. M. Hilber, et al., "Improved Numerical Dissipation for Time Integration Algorithms in Structural Mechanics", *Int. J. Earthq. Eng. Str. Dyn.*, Vol. 5, No. 3, July 1977.

14. S. Timoshenko, S. Woinowsky-Krieger, *Theory of Plates and Shells*, 2d Ed., McGraw-Hill, 1959.

15. M. Carlotto, D. Casasent, "Microprocessor-Based Fiber-Optic Iterative Optical Processor", *Applied Optics*, Vol. 21, No. 1, pp. 147-152, January 1982.

16. H. J. Caufield, et al., "Optical Implementation of Systolic Array Processing", *Optics Communications*, Vol. 40, No. 2, pp. 86-90, December 1981.

17. D. Casasent, "Acousto-Optic Transducers in Iterative Optical Vector-Matrix Processors", *Applied Optics*, Vol. 21, No. 10, pp. 1859-1865, May 1982.

18. D. Casasent, J. Jackson, C. Neuman, "Frequency-Multiplexed and Pipelined Iterative Optical Systolic Array Processors", *Applied Optics*, Vol. 22, No. 1, pp. 115-124, January 1983.

19. H. J. Whitehouse, J. Speiser, "Aspects of Signal Processing with Emphasis on Underwater Acoustics", G. Tacconi, ed., Part II, Reidel Publishing Company.

20. D. Psaltis, D. Casasent, D. Neft, M. Carlotto, "Accurate Numerical Computation by Optical Convolution", *Proc. SPIE*, Vol. 232, pp. 151-156, 1980.

21. P. S. Guilfoyle, "Systolic Acousto-Optic Binary Convolver", *Optical Engineering*, Vol. 23, No. 1, pp. 020-025, Jan/Feb 1984.

22. S. Cartwright, "New Optical Matrix-Vector Multiplier", *Applied Optics*, Vol. 23, No. 11, pp. 1683-1684, June 1984.

23. R. A. Athale, H. Q. Hoang, J. N. Lee, "High Accuracy Matrix Multiplication with a Magnetooptic Spatial Light Modulator", *Proc. SPIE*, Vol. 431, pp. 187-193, 1983.

24. R. P. Bocker, S. R. Clayton, K. Bromley, "Electrooptical Matrix Multiplication Using the Twos Complement Arithmetic for Improved Accuracy", *Applied Optics*, Vol. 22, No. 13, pp. 2019-2021, July 1983.

25. D. Casasent, J. Jackson, "Fabrication Considerations for Acousto-Optic Systolic Processors", *Proc. SPIE*, Vol. 465, pp.104-112, 1984.

26. A. Ghosh, D. Casasent, "Triangular System Solutions on an Optical Systolic Processor", *Applied Optics*, Vol. 22, No. 12, pp. 1795-1796, June 1983.

27. D. Casasent, A. Ghosh, "LU and Cholesky Decomposition on an Optical Systolic Array Processor", *Optics Communications*, Vol. 46, No. 5-6, pp. 270-273, July 1983.

28. D. Casasent, A. Ghosh, "Direct and Implicit Optical Matrix-Vector Algorithms", *Applied Optics*, Vol. 22, No. 22, pp. 3572-3578, November 1983.

29. H. T. Kung, in I. S. Duff and G. W. Stewart, eds., *Sparse Matrix Proceedings*, 1978, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

30. J. M. Spieser, H. J. Whitehouse, in Real Time Signal Processing IV, T. F. Tao, ed., *Proc. SPIE*, Vol. 298, No. 2, 1982.

31. Special Issue on Optical Computing, *Proc. IEEE*, Vol. 72, No. 7, pp. 753-992, July 1984.

32. R. P. Tewarson, *Sparse Matrices*, Academic Press, 1973.

33. R. H. Dodds Jr., L. A. Lopez, "Substructuring in Linear and Nonlinear Anaylsis", ASCE Conference on Computing in Civil Engineering, Atlanta, Ga., August 1980.

34. S. W. Bostic, "Solution of a Tridiagonal System of Equations on the Finite Element Machine", NASA Technical Memorandum 85710, March 1984.

35. D. Casasent, A. Ghosh, C. Neuman, "Direct and Indirect Optical Solutions to Linear Algebraic Equations: Error Source Modeling", *Proc. SPIE*, Vol. 431, pp. 201-208, August 1983.

36. P. Kellman, et al., "Integrating Acousto-Optic Channelized Receivers", *Proc. IEEE.*, Vol. 69, No. 1, pp. 93-100, January 1981.

37. A. Ghosh, "Performance Evaluation of Optical Linear Algebra Processors", Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, Department of Electrical and Computer Engineering, April 1984.

38. M. Tobey, et al., eds., *Operational Amplifiers: Design and Applications*, McGraw-Hill, 1971.

39. International Mathematical and Statistical Libraries, Inc., *IMSL Library Reference Manual*, 9th ed., 7500 Bellaire Blvd., Houston, Texas, 1982.

40. D. Casasent, A. Ghosh, C P. Neuman, "Iterative Solutions to Nonlinear Matrix Equations Using a Fixed Number of Steps", *Proc. SPIE*, Vol. 495, August 1984.

I. Elemental Stiffness Matrix

The following page contains the 12 by 12 elemental stiffness matrix $[K_e]$ for the case study detailed in Chapter 3. The corresponding DOF vector and equivalent elemental load vector are given in (3.2). A factor of 10^6 has been factored out of the matrix.

Elemental Stiffness Matrix

0.015	0.136	-0.056	-0.012	0.127	0.001	0.000	0.049	-0.042	-0.013	0.059	-0.013	
0.136	2.094	-0.321	-0.127	0.958	0.000	0.049	0.870	0.000	-0.058	0.524	0.000	
-0.056	-0.321	1.145	0.001	0.000	0.173	0.042	0.000	0.373	0.013	0.000	0.286	
-0.012	-0.127	0.001	0.015	-0.136	-0.056	-0.003	-0.058	-0.013	0.000	-0.049	-0.042	
0.127	0.958	0.000	-0.136	2.094	0.321	0.058	0.524	0.000	-0.049	0.870	0.000	
0.001	0.000	0.173	-0.056	0.321	1.145	0.013	0.000	0.286	0.042	0.000	0.373	
0.000	0.049	0.042	-0.003	0.058	0.013	0.015	0.136	0.036	-0.012	0.127	-0.001	
0.049	0.870	0.000	-0.058	0.524	0.000	0.136	2.094	0.321	-0.127	0.958	0.000	
-0.042	0.000	0.373	-0.013	0.000	0.286	0.036	0.321	1.145	-0.001	0.000	0.173	
-0.003	-0.058	0.013	0.000	-0.049	0.042	-0.012	-0.127	-0.001	0.015	-0.136	0.056	
0.058	0.524	0.000	-0.049	0.870	0.000	0.127	0.958	0.000	-0.136	2.094	-0.321	
-0.013	0.000	0.286	-0.042	0.000	0.373	-0.001	0.000	0.173	0.056	-0.321	1.145	

II. Structure Stiffness Matrix

The following pages contain the values in the 45 by 45 structure stiffness matrix for the case study detailed in Chapter 3. The DOF vector for the matrix is given in (3.13), and the load vector corresponds to it exactly (i.e. it is a vector of the loads for each DOF in equation 3.13, in the same order). A factor of 10^6 has been factored out of the matrix.

The matrix is given in 9 by 9 blocks, each containing the 9 matrix values corresponding to the coupling between two connected nodes in the structure. Only the lower diagonal non-zero blocks are given (since the matrix is symmetric and large). The rows and columns corresponding to the location of each 9 by 9 block in the stiffness matrix are listed above each block.

Rows 1 - 3, Columns 1 - 3

0.015	0.136	-0.056
0.136	2.094	-0.321
-0.056	-0.321	1.145

Rows 4 - 6, Columns 1 - 3

0.000	0.049	0.042
0.049	0.870	0.000
-0.042	0.000	0.373

Rows 4 - 6, Columns 4 - 6

0.030	0.272	0.000
0.272	4.189	0.000
0.000	0.000	2.289

Rows 7 - 9, Columns 4 - 6

0.000	0.049	0.042
0.049	0.870	0.000
-0.042	0.000	0.373

Rows 7 - 9, Columns 7 - 9

0.015	0.136	0.056
0.136	2.094	0.321
0.056	0.321	1.145

Rows 10 - 12, Columns 1 - 3

-0.012	-0.127	0.001
0.127	0.958	0.000
0.001	0.000	0.173

Rows 10 - 12, Columns 4 - 6

-0.003	-0.058	-0.013
0.058	0.524	0.000
0.013	0.000	0.286

Rows 10 - 12, Columns 10 - 12

0.030	0.000	-0.111
0.000	4.189	0.000
-0.111	0.000	2.289

Rows 13 - 15, Columns 1 - 3

-0.003	-0.058	0.013
0.058	0.524	0.000
-0.013	0.000	0.286

Rows 13 - 15, Columns 4 - 6

-0.024	-0.254	0.000
0.254	1.917	0.000
0.000	0.000	0.345

Rows 13 - 15, Columns 7 - 9

-0.003	-0.058	-0.013
0.058	0.524	0.000
0.013	0.000	0.286

Rows 13 - 15, Columns 10 - 12

0.000	0.000	0.084
0.000	1.739	0.000
-0.084	0.000	0.745

Rows 13 - 15, Columns 13 - 15

0 061	0. 000	0. 000
0 000	8. 378	0. 000
0 000	0 000	4. 578

Rows 16 - 18, Columns 4 - 6

-0. 003	-0 058	0. 013
0. 058	0. 524	0. 000
-0. 013	0. 000	0 286

Rows 16 - 18, Columns 7 - 9

-0. 012	-0. 127	-0 001
0. 127	0. 958	0. 000
-0 001	0. 000	0. 173

Rows 16 - 18, Columns 13 - 15

0. 000	0. 000	0. 084
0. 000	1. 739	0 000
-0. 084	0. 000	0. 745

Rows 16 - 18, Columns 16 - 18

0. 030	0. 000	0. 111
0. 000	4. 189	0. 000
0. 111	0. 000	2. 289

Rows 19 - 21, Columns 10 - 12

-0 012	-0. 127	0. 001
0. 127	0. 958	0. 000
0. 001	0. 000	0 173

Rows 19 - 21, Columns 13 - 15

-0 003	-0. 058	-0 013
0. 058	0. 524	0. 000
0. 013	0. 000	0. 286

Rows 19 - 21, Columns 19 - 21

0. 030	0. 000	-0. 111
0. 000	4. 189	0. 000
-0. 111	0. 000	2. 289

Rows 22 - 24, Columns 10 - 12

-0 003	-0. 058	0 013
0 058	0. 524	0 000
-0 013	0. 000	0 286

Rows 22 - 24, Columns 13 - 15

-0 024	-0. 254	0. 000
0 254	1. 917	0. 000
0. 000	0. 000	0. 345

Rows 22 - 24, Columns 16 - 18

-0. 003	-0. 058	-0. 013
0. 058	0. 524	0. 000
0. 013	0. 000	0. 286

Rows 22 - 24, Columns 19 - 21

0. 000	0. 000	0 084
0. 000	1. 739	0. 000
-0. 084	0. 000	0. 745

Rows 22 - 24, Columns 22 - 24

0 061	0. 000	0. 000
0. 000	9. 378	0. 000
0 000	0. 000	4. 578

Rows 25 - 27, Columns 13 - 15

-0 003	-0. 058	0. 013
0. 058	0. 524	0. 000
-0. 013	0. 000	0. 286

Rows 25 - 27, Columns 16 - 18

-0. 012	-0. 127	-0. 001
0. 127	0. 958	0. 000
-0. 001	0. 000	0. 173

Rows 25 - 27, Columns 22 - 24

0. 000	0. 000	0. 084
0. 000	1. 739	0. 000
-0. 084	0. 000	0. 745

Rows 25 - 27, Columns 25 - 27

0 030	0. 000	0 111
0 000	4 189	0 000
0 111	0 000	2. 289

Rows 28 - 30, Columns 19 - 21

-0 012	-0 127	0 001
0 127	0 958	0 000
0 001	0 000	0 173

Rows 28 - 30, Columns 22 - 24

-0. 003	-0. 058	-0 013
0 058	0. 524	0 000
0. 013	0. 000	0 286

Rows 28 - 30, Columns 28 - 30

0 030	0. 000	-0. 111
0. 000	4. 189	0. 000
-0 111	0. 000	2 289

Rows 31 - 33, Columns 19 - 21

-0 003	-0. 058	0 013
0 058	0. 524	0. 000
-0 013	0 000	0 286

Rows 31 - 33, Columns 22 - 24

-0 024	--0. 254	0 00_
0. 254	1. 917	0 000
0. 000	0. 000	0. 345

Rows 31 - 33, Columns 25 - 27

--0 003	-0. 058	-0. 013
0. 058	0. 524	0. 000
0. 013	0. 000	0. 286

Rows 31 - 33, Columns 28 - 30

0. 000	0. 000	0. 084
0. 000	1. 739	0. 000
--0. 084	0. 000	0 745

Rows 31 - 33, Columns 31 - 33

0 061	0 000	0 000
0 000	8 378	0. 000
0 000	0 000	4 576

Rows 34 - 36, Columns 22 - 24

-0 003	-0 058	0 013
0 058	0. 524	0 000
-0 013	0 000	0. 286

Rows 34 - 36, Columns 25 - 27

-0 012	-0 127	-0 001
0 127	0. 958	0. 000
-0 001	0. 000	0. 173

Rows 34 - 36, Columns 31 - 33

0. 000	0 000	0 084
0. 000	1 739	0. 000
-0. 084	0. 000	0. 745

Rows 34 - 36, Columns 34 - 36

0 030	0 000	0 111
0 000	4. 189	0. 000
0. 111	0. 000	2. 285

Rows 37 - 39, Columns 28 - 30

-0. 012	-0 127	0. 001
0 127	0. 958	0 000
0. 001	0. 000	0. 173

Rows 37 - 39, Columns 31 - 33

-0. 003	-0. 058	-0. 013
0. 058	0. 524	0. 000
0. 013	0. 000	0. 286

Rows 37 - 39, Columns 37 - 39

0 015	-0. 136	-0. 056
-0 136	2. 094	0. 321
-0. 056	0. 321	1 145

Rows 40 - 42, Columns 28 - 30

-0.003	-0.058	0.013
0.058	0.524	0.000
-0.013	0.000	0.286

Rows 40 - 42, Columns 31 - 33

-0.024	-0.254	0.000
0.254	1.917	0.000
0.000	0.000	0.345

Rows 40 - 42, Columns 34 - 36

-0.003	-0.058	-0.013
0.058	0.524	0.000
0.013	0.000	0.286

Rows 40 - 42, Columns 37 - 39

0.000	-0.049	0.042
-0.049	0.870	0.000
-0.042	0.000	0.373

Rows 40 - 42, Columns 40 - 42

0.030	-0.272	0.000
-0.272	4.189	0.000
0.000	0.000	2.289

Rows 43 - 45, Columns 31 - 33

-0.003	-0.058	0.013
0.058	0.524	0.000
-0.013	0.000	0.286

Rows 43 - 45, Columns 34 - 36

-0.012	-0.127	-0.001
0.127	0.958	0.000
-0.001	0.000	0.173

Rows 43 - 45, Columns 40 - 42

0.000	-0.049	0.042
-0.049	0.870	0.000
-0.042	0.000	0.373

Rows 43 - 45, Columns 43 - 45

0 015	-0 136	0 056
-0 136	2 094	-0 321
0 056	-0 321	1 145